CUPS Software Programmers Manual

CUPS−SPM−1.1.6

Easy Software Products

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Preface

This software programmers manual provides software programming information for the Common UNIX Printing System ("CUPS") Version 1.1.6.

System Overview

CUPS provides a portable printing layer for UNIX®−based operating systems. It has been developed by Easy Software Products to promote a standard printing solution for all UNIX vendors and users. CUPS provides the System V and Berkeley command−line interfaces.

CUPS uses the Internet Printing Protocol ("IPP") as the basis for managing print jobs and queues. The Line Printer Daemon ("LPD") Server Message Block ("SMB"), and AppSocket (a.k.a. JetDirect) protocols are also supported with reduced functionality. CUPS adds network printer browsing and PostScript Printer Description ("PPD") based printing options to support real−world printing under UNIX.

CUPS also includes a customized version of GNU Ghostscript (currently based off GNU Ghostscript 5.50) and an image file RIP that are used to support non−PostScript printers. Sample drivers for HP and EPSON printers are included that use these filters.

Document Overview

This software programmers manual is organized into the following sections:

Notation Conventions

Various font and syntax conventions are used in this guide. Examples and their meanings and uses are explained below:

Example	Description
`lpstat` `lpstat(1)`	The names of commands; the first mention of a command or function in a chapter is followed by a manual page section number.
/var */usr/share/cups/data/testprint.ps*	File and directory names.
`Request ID is Printer-123`	Screen output.
`lp -d printer filename ENTER`	Literal user input; special keys like **`ENTER`** are in ALL CAPS.
12.3	Numbers in the text are written using the period (.) to indicate the decimal point.

Abbreviations

The following abbreviations are used throughout this manual:

kb
: Kilobytes, or 1024 bytes

Mb
: Megabytes, or 1048576 bytes

Gb
: Gigabytes, or 1073741824 bytes

Other References

CUPS Software Administrators Manual
: An administration guide for the CUPS software.

CUPS Software Users Manual
: An end−user guide for using the CUPS software.

1 − Printing System Overview

This chapter provides an overview of how the Common UNIX Printing System works.

The Printing Problem

For years *the printing problem* has plagued UNIX. Unlike Microsoft® Windows® or Mac OS, UNIX has no standard interface or system in place for supporting printers. Among the solutions currently available, the Berkeley and System V printing systems are the most prevalent.

These printing systems support line printers (text only) or PostScript printers (text and graphics), and with some coaxing they can be made to support a full range of printers and file formats. However, because each varient of the UNIX operating system uses a different printing system than the next developing printer drivers for a wide range of printers and operating systems is extremely difficult. That combined with the limited volume of customers for each UNIX varient has forced most printer vendors to give up supporting UNIX entirely.

CUPS is designed to eliminate *the printing problem*. One common printing system can be used by all UNIX varients to support the printing needs of users. Printer vendors can use its modular filter interface to develop a single driver program that supports a wide range of file formats with little or no effort. Since CUPS provides both the System V and Berkeley printing commands, users (and applications) can reap the benefits of this new technology with no changes.

The Technology

CUPS is based upon an emerging Internet standard called the Internet Printing Protocol. IPP has been embraced by dozens of printer and printer server manufacturers and is supported by Microsoft Windows 2000.

IPP defines a standard protocol for printing as well as managing print jobs and printer options like media size, resolution, and so forth. Like all IP−based protocols, IPP can be used locally or over the Internet to printers hundreds or thousands of miles away. Unlike other protocols, however, IPP also supports access control, authentication, and encryption, making it a much more capable and secure printing solution than older ones.

IPP is layered on top of the Hyper−Text Transport Protocol ("HTTP") which is the basis of web servers on the Internet. This allows users to view documentation, check status information on a printer or server, and manage their printers, classes, and jobs using their web browser.

CUPS provides a complete IPP/1.1 based printing system that provides Basic, Digest, and local certificate authentication and user, domain, or IP−based access control. TLS encryption will be available in future versions of CUPS.

Jobs

Each file or set of files that is submitted for printing is called a *job*. Jobs are identified by a unique number starting at 1 and are assigned to a particular destination, usually a printer. Jobs can also have options associated with them such as media size, number of copies, and priority.

Classes

CUPS supports collections of printers known as *classes*. Jobs sent to a class are forwarded to the first available printer in the class.

Filters

Filters allow a user or application to print many types of files without extra effort. Print jobs sent to a CUPS server are filtered before sending them to a printer. Some filters convert job files to different formats that the printer can understand. Others perform page selection and ordering tasks.

CUPS provides filters for printing many types of image files, HP−GL/2 files, PDF files, and text files. CUPS also supplies PostScript and image file Raster Image Processor ("RIP") filters that convert PostScript or image files into bitmaps that can be sent to a raster printer.

Backends

Backends perform the most important task of all − they send the filtered print data to the printer.

CUPS provides backends for printing over parallel, serial, and USB ports, and over the network via the IPP, JetDirect (AppSocket), and Line Printer Daemon ("LPD") protocols. Additional backends are available in network service packages such as the SMB backend included with the popular SAMBA software.

Backends are also used to determine the available devices. On startup each backend is asked for a list of devices it supports, and any information that is available. This allows the parallel backend to tell CUPS that an EPSON Stylus Color 600 printer is attached to parallel port 1, for example.

Printer Drivers

Printer drivers in CUPS consist of one of more filters specific to a printer. CUPS includes sample printer drivers for Hewlett−Packard LaserJet and DeskJet printers and EPSON 9−pin, 24−pin, Stylus Color, and Stylus Photo printers. While these drivers do not generate optimal output for the different printer models, they do provide basic printing and demonstrate how you can write your own printer drivers and incorporate them into CUPS.

Networking

Printers and classes on the local system are automatically shared with other systems on the network. This allows you to setup one system to print to a printer and use this system as a printer server or spool host for all of the others. Users may then select a local printer by name or a remote printer using "name@server".

CUPS also provides *implicit classes*, which are collections of printers and/or classes with the same name. This allows you to setup multiple servers pointing to the same physical network printer, for example, so that you aren't relying on a single system for printing. Because this also works with printer classes, you can setup multiple servers and printers and never worry about a single point of failure unless all of the printers and servers go down!

2 − The CUPS API

This chapter describes the CUPS Application Programmers Interface ("API").

The CUPS API Library

The CUPS library provides a whole collection of interfaces needed to support the internal needs of the CUPS software as well as the needs of applications, filters, printer drivers, and backends.

Unlike the rest of CUPS, the CUPS API library is provided under the GNU Library General Public License. This means that you can use the CUPS API library in both proprietary and open−source programs.

Programs that use the CUPS API library typically will include the `<cups/cups.h>` header file:

```
#include <cups/cups.h>

...

jobid = cupsPrintFile("myprinter", "filename.ps", "title",
                      num_options, options);
```

Use the `−lcups` compiler option when linking to the CUPS API library:

```
cc −o program program.c −lcups ENTER
```

Additional options and libraries may be required depending on the operating system and the location of the CUPS API library.

Detecting the CUPS API Library in GNU Autoconf

GNU autoconf is a popular configuration tool used by many programs. Add the following lines to your *configure.in file to check for the CUPS API library in your configuration script:*

```
AC_CHECK_LIB(socket,socket,
if test "$uname" != "IRIX"; then
        LIBS="-lsocket $LIBS"
else
        echo "Not using -lsocket since you are running IRIX."
fi)
AC_CHECK_LIB(nsl,gethostbyaddr,
if test "$uname" != "IRIX"; then
        LIBS="-lnsl $LIBS"
else
        echo "Not using -lnsl since you are running IRIX."
fi)

AC_CHECK_LIB(cups,httpConnect)
```

Printing Services

The CUPS API library provides some basic printing services for applications that need to print files.

Include Files

The include file used by all of these functions is `<cups/cups.h>`:

```
#include <cups/cups.h>
```

Printing a File

The CUPS API provides two functions for printing files. The first is `cupsPrintFile` which prints a single named file:

```
#include <cups/cups.h>

...

int jobid;

...

jobid = cupsPrintFile("name", "filename", "title", 0, NULL);
```

The `name` string is the name of the printer or class to print to. The `filename` string is the name of the file to print. The `title` string is the name of the print job, e.g. "Acme Word Document".

The return value is a unique ID number for the print job or 0 if there was an error.

Printing Multiple Files

The second printing function is `cupsPrintFiles`:

```
#include <cups/cups.h>

...

int        jobid;
int        num_files;
const char *files[100];
...

jobid = cupsPrintFiles("name", num_files, files, "title", 0, NULL);
```

Instead of passing a filename string as with `cupsPrintFile()`, you pass a file count (`num_files`) and filename pointer array (`files`) for each file that you want to print.

As with `cupsPrintFile()`, the return value is a unique ID for the print job.

Cancelling Jobs

The `cupsCancelJob()` function cancels a queued print job:

```
#include <cups/cups.h>

...

int jobid;
int status;
...

status = cupsCancelJob("name", jobid);
```

The `name` string specifies the destination and is used to determine the server to send the request to. The `jobid` value is the integer returned from a previous `cupsPrintFile()` or `cupsPrintFiles()` call.

`cupsCancelJob()` returns `1` if the job was successfully cancelled and `0` if there was an error.

Getting the Available Printers and Classes

The `cupsGetDests()` function can be used to get a list of the available printers, classes, and instances that a user has defined:

```
#include <cups/cups.h>

...

int         num_dests;
cups_dest_t *dests;

...

num_dests = cupsGetDests(&dests);
```

Each destination is stored in a `cups_dest_t` structure which defines the printer or class name, the instance name (if any), if it is the default destination, and the default options the user has defined for the destination:

```
typedef struct               /**** Destination ****/
{
```

```
  char          *name,       /* Printer or class name */
                *instance;   /* Local instance name or NULL */
  int           is_default;  /* Is this printer the default? */
  int           num_options; /* Number of options */
  cups_option_t *options;    /* Options */
} cups_dest_t;
```

The destinations are sorted by name and instance for your convenience. Once you have the list of available destinations, you can lookup a specific destination using the `cupsGetDest()` function:

```
#include <cups/cups.h>

...

int         num_dests;
cups_dest_t *dests;
cups_dest_t *mydest;

...

mydest = cupsGetDest("name", "instance", num_dests, dests);
```

The `name` string is the printer or class name. You can pass a value of `NULL` to get the default destination.

The `instance` string is the user−defined instance name. Pass `NULL` to select the default instance, e.g. "name" instead of "name/instance".

Printing with Options

All of the previous printing examples have passed `0` and `NULL` for the last two arguments to the `cupsPrintFile()` and `cupsPrintFiles()` functions. These last two arguments are the number of options and a pointer to the option array:

```
int cupsPrintFile(const char *name, const char *filename, const char *title,
                  int num_options, cups_option_t *options);
int cupsPrintFiles(const char *name, int num_files, const char **files,
                   const char *title, int num_options,
                   cups_option_t *options);
```

The `cups_option_t` structure holds each option and its value. These are converted as needed and passed to the CUPS server when printing a file.

The simplest way of handling options is to use the `num_options` and `options` members of the `cups_dest_t` structure described earlier:

```
#include <cups/cups.h>

...

int         jobid;
int         num_dests;
cups_dest_t *dests;
cups_dest_t *mydest;

...

mydest = cupsGetDest("name", "instance", num_dests, dests);
```

```
jobid  = cupsPrintFile(mydest->name, "filename", "title",
                       mydest->num_options, mydest->options);
```

This effectively uses the options a user has previous selected without a lot of code.

Setting Printer Options

Options can also be set by your program using the `cupsAddOption()` function:

```
#include <cups/cups.h>

...

int           num_options;
cups_option_t *options;

...

num_options = 0;
options     = NULL;

...

num_options = cupsAddOption("name", "value", num_options, &options);
num_options = cupsAddOption("name", "value", num_options, &options);
num_options = cupsAddOption("name", "value", num_options, &options);
num_options = cupsAddOption("name", "value", num_options, &options);
```

The `name` string is the name of the option, and the `value` string is the value for that option.

Each call to `cupsAddOption()` returns the new number of options. Since adding two options with the same name overwrites the first value with the second, do not assume that calling `cupsAddOptions()` 20 times will result in 20 options.

Call `cupsFreeOptions` once you are done using the options:

```
#include <cups/cups.h>

...

int           num_options;
cups_option_t *options;

...

cupsFreeOptions(num_options, options);
```

Getting Errors

If any of the CUPS API printing functions returns an error, the reason for that error can be found by calling `cupsLastError()` and `cupsErrorString()`. `cupsLastError()` returns the last IPP error code that was encountered. `cupsErrorString()` converts the error code to a localized message string suitable for presentation to the user:

```
#include <cups/cups.h>
```

```
...

int jobid;

...

if (jobid == 0)
  puts(cupsErrorString(cupsLastError()));
```

Passwords and Authentication

CUPS supports authentication of any request, including submission of print jobs. The default mechanism for getting the username and password is to use the login user and a password from the console.

To support other types of applications, in particular Graphical User Interfaces ("GUIs"), the CUPS API provides functions to set the default username and to register a callback function that returns a password string.

The `cupsSetPasswordCB()` function is used to set a password callback in your program. Only one function can be used at any time.

The `cupsSetUser()` function sets the current username for authentication. This function can be called by your password callback function to change the current username as needed.

The following example shows a simple password callback that gets a username and password from the user:

```
#include <cups/cups.h>

const char *
my_password_cb(const char *prompt)
{
  char  user[65];


  puts(prompt);

 /* Get a username from the user */
  printf("Username: ");
  if (fgets(user, sizeof(user), stdin) == NULL)
    return (NULL);

 /* Strip the newline from the string and set the user */
  user[strlen(user) - 1] = '\0';

  cupsSetUser(user);

 /* Use getpass() to ask for the password... */
  return (getpass("Password: "));
}

...

cupsSetPasswordCB(my_password_cb);
```

Similarly, a GUI interface could display the prompt string in a window with input fields for the username and password. The username should probably default to the value of `cupsUser()` to make things easier on the

user.

PPD Services

CUPS includes functions to access and manipulate PostScript Printer Description ("PPD") files that are used with the printer drivers in CUPS.

Each PPD file enumerates the available features provided by a printer, including conflict information for specific options (e.g. can't duplex output on envelopes.)

Include Files

Include the `<cups/ppd.h>` header file to use the PPD functions:

```
#include <cups/ppd.h>
```

This header file is also included by the `<cups/cups.h>` header file.

Getting a PPD File for a Printer

The `cupsGetPPD()` function retrieves the PPD file for the named printer or class:

```
#include <cups/cups.h>

...

const char *filename;

filename = cupsGetPPD("name");
```

The `name` string is the name of the printer or class, including the remote server name as appropriate (e.g. "printer@server".)

The return value is a pointer to a filename in static storage; this value is overwritten with each call to `cupsGetPPD()`. If the printer or class does not exist, a `NULL` pointer will be returned.

Loading a PPD File

The `ppdOpenFile()` function "opens" a PPD file and loads it into memory:

```
#include <cups/ppd.h>

...

ppd_file_t *ppd;

ppd = ppdOpenFile("filename");
```

The `filename` string is the name of the file to load, such as the value returned by the `cupsGetPPD()` function.

The return value is a pointer to a structure describing the contents of the PPD file or NULL if the PPD file could not be read.

Freeing PPD File Information

Once you are done using a PPD file, call the `ppdClose()` function to free all memory that has been used:

```
#include <cups/ppd.h>

...

ppd_file_t *ppd;

...

ppdClose(ppd);
```

The PPD File Structure

Each PPD file contains a number of capability attributes, printer options, and conflict definitions. The page size options also include the physical margins for the printer and the minimum and maximum sizes for the printer. All of this information is stored in the `ppd_file_t` structure.

Capabilities

Each PPD file contains a number of informational attributes that describe the capabilities of the printer. These are provided in the `ppd_file_t` structure in the following members:

Member	Type	Description
`accurate_screens`	`int`	1 = supports accurate screens
`color_device`	`int`	1 = color device
`colorspace`	`ppd_cs_t`	Default colorspace: PPD_CS_CMYK, PPD_CS_CMY, PPD_CS_GRAY, PPD_CS_RGB, PPD_CS_RGBK, PPD_CS_N
`contone_only`	`int`	1 = printer is continuous tone only
`num_emulations` `emulations`	`int` `ppd_emul_t *`	Emulations supported by the printer
`flip_duplex`	`int`	1 = need to flip odd pages when duplexing
`num_fonts` `fonts`	`int` `char **`	The fonts available on the printer.
	`char *`	

jcl_begin jcl_ps jcl_end		Job Control Language commands for PostScript output
landscape	int	Landscape orientation, −90 or 90 degrees
lang_encoding	char *	The character used for the option strings
lang_version	char *	The language used for the options strings (English, French, etc.)
language_level	int	PostScript language level, 1 to 3
manual_copies	int	1 = Copies are done manually
model_number	int	Driver−specific model number.
patches	char *	Patch commands to send to the printer
manufacturer	char *	The Manufacturer attribute from the PPD file, if any
modelname	char *	The ModelName attribute from the PPD file
nickname	char *	The NickName attribute from the PPD file, if any
product	char *	The Product attribute from the PPD file, if any
shortnickname	char *	The ShortNickName attribute from the PPD file, if any
throughput	int	Number of pages per minute
ttrasterizer	char *	The TruType font rasterizer (Type42)
variable_sizes	int	1 = supports variable sizes

Options and Groups

PPD files support multiple options, which are stored in `ppd_option_t` and `ppd_choice_t` structures by the PPD functions.

Each option in turn is associated with a group stored in the `ppd_group_t` structure. Groups can be specified in the PPD file; if an option is not associated with a group then it is put in a "General" or "Extra"

group depending on the option.

Groups can also have sub−groups; CUPS currently limits the depth of sub−groups to 1 level to reduce programming complexity.

Conflicts

PPD files support specification of conflict conditions between different options. Conflicts are stored in `ppd_conflict_t` structures which specify the options that conflict with each other.

Page Sizes

PPD files specify all of the available pages sizes and the physical margins associated with them. These sizes are stored in `ppd_size_t` structures and are available in the `num_sizes` and `sizes` members of the `ppd_file_t` structure. You can lookup a particular page size with the `ppdPageWidth()`, `ppdPageLength()`, and `ppdPageSize()` functions:

```
#include <cups/ppd.h>

...

ppd_file_t *ppd;
ppd_size_t *size;
float      width;
float      length;

...

size   = ppdPageSize(ppd, "size");
width  = ppdPageWidth(ppd, "size");
length = ppdPageLength(ppd, "size");
```

The `size` string is the named page size option. The width and length are in points; there are 72 points per inch. The `ppd_size_t` structure contains the width, length, and margin information:

```
typedef struct    /**** Page Sizes ****/
{
  int   marked;   /* Page size selected? */
  char  name[41]; /* Media size option */
  float width,    /* Width of media in points */
        length,   /* Length of media in points */
        left,     /* Left printable margin in points */
        bottom,   /* Bottom printable margin in points */
        right,    /* Right printable margin in points */
        top;      /* Top printable margin in points */
} ppd_size_t;
```

Custom Page Sizes

Besides the standard page sizes listed in a PPD file, some printers support variable or custom page sizes. If `variables_sizes` is non−zero, the `custom_min`, `custom_max`, and `custom_margins` members of the `ppd_file_t` structure define the limits of the variable sizes.

To get the resulting media size, use a page size string of `Custom.`*`width`*`x`*`length`*, where `width` and `length` are integer values in points:

```
Custom.612x792   [8.5 inches wide, 11 inches long]
Custom.1224x792  [17 inches wide, 11 inches long]
```

Marking Options

Before marking any user−defined options, call the `ppdMarkDefaults()` function to mark the default options from the PPD file:

```
#include <cups/ppd.h>

...

ppd_file_t *ppd;

...

ppdMarkDefaults(ppd);
```

Then call the `ppdMarkOption()` function to mark individual options:

```
#include <cups/ppd.h>

...

ppd_file_t *ppd;
int        conflicts;

...

conflicts = ppdMarkOption(ppd, "name", "value");
```

The `name` and `value` strings choose a particular option and choice, respectively. The return value is 0 if there are not conflicts created by the selection.

CUPS also provides a convenience function for marking all options in the `cups_option_t` structure:

```
#include <cups/cups.h>

...

ppd_file_t    *ppd;
int           num_options;
cups_option_t *options;
int           conflicts;

...

conflicts = cupsMarkOptions(ppd, num_options, options);
```

The `cupsMarkOptions()` function also handles mapping the IPP job template attributes to PPD options. The return value is the number of conflicts present.

Checking for Conflicts

The `ppdMarkOption()` and `cupsMarkOptions()` functions return the number of conflicts with the currently marked options.

Call the `ppdConflicts()` function to get the number of conflicts after you have marked all of the options:

```
#include <cups/cups.h>

...

ppd_file_t *ppd;
int        conflicts;

...

conflicts = ppdConflicts(ppd);
```

The return value is the number of conflicting options, or 0 if there are no conflicts.

3 − Writing Filters

This chapter describes how to write a file filter for CUPS.

Overview

File filters are programs that convert from one or more MIME types to another type. Filters use a common command−line and environment interface that allows them to be joined as needed to print files to any type of printer.

Security Considerations

Filters are normally run as a non−priviledged user, so the major security consideration is resource utilization − filters should not depend on unlimited amounts of memory and disk space.

Users and Groups

The default CUPS configuration runs filters as user "lp" and group "other".

Temporary Files

Temporary files should be created in the directory specified by the "TMPDIR" environment variable. The `cupsTempFile()` function can be used to safely choose temporary files in this directory.

Sending Messages to the User

The CUPS scheduler collects messages sent to the standard error file by the filter. These messages are relayed to the user based upon the scheduler `LogLevel` directive.

The type of message is determined by an initial prefix sent on each line:

- `DEBUG:` − a debug message
- `INFO:` − an informational message
- `WARNING:` − a warning message
- `ERROR:` − an error message
- `PAGE:` − a page accounting message

If the line of text does not begin with any of the above prefixes, it is treated as a debug message. Text following the prefix is copied to the `printer-state-message` attribute for the printer, and also added to the *error_log* unless it is an informational or page accounting message.

Page Accounting

Page accounting messages are used to inform the server when one or more pages are printed. Each line has the form:

```
PAGE: page-number copy-count
```

The *page-number* field is the current page number, starting at 1. The *copy-count* field specifies the number of copies of that page that was produced.

Page account messages are added to the *page_log* file and cause the `job-sheets-completed` attribute to be updated for the job.

Command-Line Arguments

Every filter accepts exactly 6 or 7 command-line arguments:

```
printer job user title copies options [filename]
```

- `printer` − The name of the printer queue (normally this is the name of the program being run)
- `job` − The numeric job ID for the job being printed
- `user` − The string from the `originating-user-name` attribute
- `title` − The string from the `job-name` attribute
- `copies` − The numeric value from the `number-copies` attribute
- `options` − String representations of the job template attributes, separated by spaces. Boolean attributes are provided as "name" for true values and "noname" for false values. All other attributes are provided as "name=value" for single-valued attributes and "name=value1,value2,...,valueN" for set attributes
- `filename` − The request file

The *filename* argument is only provided to the first filter in the chain; all filters **must** be prepared to read the print file from the standard input if the *filename* argument is omitted.

Copy Generation

The *copies* argument specifies the number of copies to produce of the input file. In general, you should only generate copies if the *filename* argument is supplied. The only exception to this are filters that produce device−independent PostScript output (without any printer commands from the printer's PPD file), since the PostScript filter `pstops` is responsible for copy generation.

Environment Variables

Every filter receives a fixed set of environment variables that can be used by the filter:

- `CHARSET` − The character set used by the client for this print file
- `CONTENT_TYPE` − The original document type, such as "application/postscript"
- `CUPS_DATADIR` − The location of CUPS data files
- `CUPS_SERVERROOT` − The location of CUPS configuration files
- `DEVICE_URI` − The output device URI
- `LANG` − The language used by the client for this print file
- `PATH` − The execution path exported to the filter
- `PPD` − The full filename of the printer's PPD file
- `PRINTER` − The name of the printer queue
- `RIP_CACHE` − The maximum amount of memory each filter should use
- `SOFTWARE` − The name of the CUPS software, typically "CUPS/1.1"
- `TZ` − The local timezone
- `USER` − The name of the current user

Dissecting the HP−GL/2 Filter

The HP−GL/2 filter (`hpgltops`) provided with CUPS is a complex program that converts HP−GL/2 files into device−independent PostScript output. Since it produces device−independent PostScript output, it does not need to handle copy generation or writing printer options from the printer's PPD file.

Initializing the Filter

The first task of any filter is to ensure that the correct number of command−line arguments are present:

```
if (argc < 6 || argc > 7)
{
  fputs("ERROR: hpgltops job−id user title copies options [file]\n", stderr);
  return (1);
}
```

After this you open the print file or read from the standard input as needed:

```
FILE *fp;

/*
 * If we have 7 arguments, print the file named on the command−line.
 * Otherwise, send stdin instead...
 */

if (argc == 6)
  fp = stdin;
```

```
else
{
 /*
  * Try to open the print file...
  */

  if ((fp = fopen(argv[6], "rb")) == NULL)
  {
    perror("ERROR: unable to open print file - ");
    return (1);
  }
}
```

Once the print file has been opened, options can be processed using the `cupsParseOptions()` and `cupsGetOption()` functions:

```
int           num_options;
cups_option_t *options;
const char    *val;

/*
 * Process command-line options and write the prolog...
 */

options     = NULL;
num_options = cupsParseOptions(argv[5], 0,

if ((val = cupsGetOption("blackplot", num_options, options)) != NULL)
  shading = 0;

if ((val = cupsGetOption("fitplot", num_options, options)) != NULL)
  FitPlot = 1;

if ((val = cupsGetOption("penwidth", num_options, options)) != NULL)
  PenWidth = (float)atoi(val) * 0.001f;
```

After the options have been processed, the filter writes PostScript code to the standard output based on the print file, closes the print file (as needed), and returns 0 to the scheduler.

PostScript Output

Filters that produce PostScript output must generate output conforming to the Adobe Document Structuring Conventions, 3.0. In general this means the beginning of each file must begin with:

```
%!PS-Adobe-3.0
%%BoundingBox: left bottom right top
%%Pages: (atend)
%%EndComments
```

The *left*, *bottom*, *right*, and *top* values are integers in points from the lower-lefthand corner of the page.

Pages must be surrounded by:

```
%%Page: number number
gsave
...
grestore
```

```
showpage
```

And the end of each file must contain:

```
%%Trailer
%%Pages: number-pages
%%EOF
```

These comments allow the PostScript filter to correctly perform page accounting, copy generation, N−up printing, and so forth.

4 − Writing Printer Drivers

This chapter discusses how to write a printer driver, which is a special filter program that converts CUPS raster data into the appropriate commands and data required for a printer.

Overview

Raster printers utilitize PPD files that specify one or more device−specific filters that handle converting print files for the printer. The simplest raster printer drivers provide a single filter that converts CUPS raster data to the printer's native format.

CUPS Raster Data

CUPS raster data (`application/vnd.cups−raster`) consists of a stream of raster page descriptions produced by one of the RIP filters, such as `pstoraster` or `imagetoraster`.

Each page of data begins with a page dictionary structure called `cups_raster_header_t`. This structure contains the colorspace, bits per color, media size, media type, hardware resolution, and so forth.

After the page dictionary comes the page data which is a full−resolution, uncompressed bitmap representing the page in the printer's output colorspace.

Page Accounting

Printer drivers must handle all page accounting. This means they must send "PAGE:" messages to the standard error file for each page (and in many cases, copy) sent to the printer.

Color Management

Printer drivers can implement their color management via the `cupsColorProfile` attributes in the PPD file or internally in the driver from a device−independent colorspace. In general, color management performed by the RIP filters is more efficient than that performed inside printer drivers.

For example, the `pstoraster` filter often only has to perform a color conversion once each time the color is used for multiple output pixels, while the raster filter must convert every pixel on the page.

Device and Bitmap Variables

Besides the standard PostScript page device dictionary variables defined in the Adobe PostScript Level 3 reference manual, the CUPS filters support additional variables that are passed in the page device dictionary header for the page and in some cases control the type of raster data that is generated:

Variable	Type	Description
cupsWidth	read−only integer	Width of bitmap in pixels
cupsHeight	read−only integer	Height of bitmap in pixels
cupsMediaType	read−write integer	Device−specific media type code
cupsBitsPerColor	read−write integer	Number of bits per color; 1, 2, 4, and 8 are currently supported
cupsBitsPerPixel	read−only integer	Number of bits per pixel; 1 to 32
cupsBytesPerLine	read−only integer	Number of bytes per line of raster graphics
cupsColorOrder	read−write enum	The order of color values in the bitmap: • `CUPS_ORDER_CHUNKED` − CMYK CMYK CMYK • `CUPS_ORDER_BANDED` − CCC MMM YYY KKK • `CUPS_ORDER_PLANAR` − CCC ... MMM ... YYY ... KKK ...
cupsColorSpace	read−write enum	The colorspace of the bitmap: • `CUPS_CSPACE_W` − White (luminance) • `CUPS_CSPACE_RGB` − Red, green, blue • `CUPS_CSPACE_RGBA` − Red, green, blue, alpha • `CUPS_CSPACE_K` − Black • `CUPS_CSPACE_CMY` − Cyan, magenta, yellow • `CUPS_CSPACE_YMC` − Yellow, magenta, cyan

		• CUPS_CSPACE_CMYK − Cyan, magenta, yellow, black • CUPS_CSPACE_YMCK − Yellow, magenta, cyan, black • CUPS_CSPACE_KCMY − Black, cyan, magenta, yellow • CUPS_CSPACE_KCMYcm − Black, cyan, magenta, yellow, light cyan, light magenta • CUPS_CSPACE_GMCK − Metallic yellow (gold), metallic magenta, metallic cyan, black • CUPS_CSPACE_GMCS − Metallic yellow (gold), metallic magenta, metallic cyan, metallic grey (silver) • CUPS_CSPACE_WHITE − White pigment (black as white pigment) • CUPS_CSPACE_GOLD − Gold foil (black as gold foil) • CUPS_CSPACE_SILVER − Silver foil (black as silver foil)
cupsCompression	read−write integer	Device−specific compression type code
cupsRowCount	read−write integer	Device−specific row count value
cupsRowFeed	read−write integer	Device−specific row feed value
cupsRowStep	read−write integer	Device−specific row step value

Bitmaps with a colorspace of CUPS_CSPACE_KCMYcm and more than 1 bit per color are transmitted to the raster driver in KCMY colorspace; the driver is responsible for producing the correct separation of normal and light cyan and magenta inks.

Dissecting the HP−PCL Driver

The HP−PCL driver provided with CUPS (`rastertohp`) converts bitmap data from the raster filters into HP−PCL commands for most PCL−compatible printers. The actual format of the raster data is controlled by the PPD file being used − *deskjet.ppd* or *laserjet.ppd*.

PPD Files

PPD files play an important part of all raster printer drivers. Options defined in the PPD file contain PostScript commands that control the raster data that is sent to the printer driver.

A typical CUPS printer driver will include `ColorModel`, `InputSlot`, `PageSize`, `PageRegion`, and `Resolution` options. Each option is shown using the standard PPD format:

```
*OpenUI *PageSize/Media Size: PickOne
*OrderDependency: 10 AnySetup *PageSize
*DefaultPageSize: Letter
*PageSize Letter/US Letter: "<<
/PageSize [612 792]
```

```
/ImagingBBox null
>> setpagedevice"
*End
*PageSize Legal/US Legal: "<<
/PageSize [612 1008]
/ImagingBBox null
>> setpagedevice"
*End
*PageSize A4/A4: "<<
/PageSize [595 842]
/ImagingBBox null
>> setpagedevice"
*End
*CloseUI: *PageSize
```

The `OpenUI` keyword specifies the new option. The first name is the option with an asterisk (*) in front of it. The first name is usually followed by a slash (/) and a human−readable version of the option name.

Every option **must** have a default value, specified using the `Default`*`Option`* keyword.

Each option begins with the option name followed by the computer and human−readable values. The PostScript commands follow these inside double quotes. PostScript commands can be provided on a single line:

```
*PageSize A4/A4: "<</PageSize[595 842]/ImagingBBox null>> setpagedevice"
```

or broken down on separate lines using the `End` keyword to terminate them:

```
*PageSize A4/A4: "<<
/PageSize [595 842]
/ImagingBBox null
>> setpagedevice"
*End
```

The choice of the two formats is usually esthetic. However, each line in a PPD file must not exceed 255 characters, so if your PostScript commands are long you may need to break them up on separate lines.

Reading Raster Data

As with any filter, your printer driver should handle raster data from a filename specified on the command−line or from the standard input. The `cupsRasterOpen()` function opens a raster stream for printing:

```
int           fd;   /* File descriptor */
cups_raster_t *ras; /* Raster stream for printing */


/*
 * Check for valid arguments...
 */

if (argc < 6 || argc > 7)
{
 /*
  * We don't have the correct number of arguments; write an error message
  * and return.
  */
```

```
  fputs("ERROR: rastertopcl job-id user title copies options [file]\n", stderr);
  return (1);
}

/*
 * Open the page stream...
 */

if (argc == 7)
{
  if ((fd = open(argv[6], O_RDONLY)) == -1)
  {
    perror("ERROR: Unable to open raster file - ");
    sleep(1);
    return (1);
  }
}
else
  fd = 0;

ras = cupsRasterOpen(fd, CUPS_RASTER_READ);
```

Once you have opened the raster stream you just need to read each page and print it:

```
cups_raster_header_t header;
int                  y;
unsigned char        data[8192];

while (cupsRasterReadHeader(ras, &header))
{
  ... initialize the printer ...
  for (y = header.cupsHeight; y > 0; y ++)
  {
    cupsRasterReadPixels(ras, data, header.cupsBytesPerLine);
    ... send raster line to printer ...
  }
}
```

After you have processed all pages, close the raster stream and return:

```
cupsRasterClose(ras);

return (0);
```

5 − Writing Backends

This chapter describes how to write a backend for CUPS. Backends communicate directly with printers and allow printer drivers and filters to send data using any type of connection transparently.

Overview

Backends are special filters that communicate with printers directly. They are treated slightly differently than filters, however, and have some unique requirements.

Security Considerations

Backends are run as the root user, so special care must be taken to avoid potential security violations. In particular, remember that a backend will be able to manipulate disk files, devices, and other resources that potentially could damage a system or printer.

Command−Line Arguments

Besides the standard filter arguments, backends are also run with no arguments to get a list of available devices. This discovery process is described later in this chapter.

Copy Generation

Like filters, backends should send multiple copies of the print file only if a filename is supplied on the command−line. Otherwise the backend should assume that the upstream filter has already added the

necessary commands or data to produce the multiple copies.

Page Accounting

Backend filters generally do not do page accounting, however they should at a minimum produce a single page message for each copy that is produced when a filename is present on the command−line. This is because the user selected "raw" printing and no other accounting information is possible.

Exclusive Access

Backends that talk to local character or block devices should open the device file in exclusive mode (`O_EXCL`) to cooperate with other printers defined for the same device.

Retries

All backends **must** retry connections to the device. This includes backends that talk to local character or block devices, as the user may define more than one printer queue pointing at the same physical device.

To prevent excess CPU utilitization, the backend should go to sleep for an amount of time between retries; the CUPS−supplied backends retry once every 30 seconds.

Dissecting the Serial Port Backend

The serial port backend provides support for serial printers. Since it does everything a good backend needs to do, it provides an excellent example of what to do.

Supporting Device Discovery

As previously noted, backends are special filter programs that talk to printer devices. Another task a backend must perform is to list the available devices it supports. The backend lists the available devices when no additioanl arguments are supplied on the command−line (i.e. just the command name...)

The serial backend lists devices by looking at serial port files in the */dev* directory, by consulting a hardware inventory (IRIX), and in some cases by trying to open the ports to see if they actually exist.

Once it finds a serial port it writes a single line for each port to the standard error file. Each line looks like this:

```
serial serial:/dev/ttyS0?baud=115200 "Unknown" "Serial Port 1"
```

The first word "serial" is the *device class*; this identifies the class of device which can be used to categorize it in user interfaces. CUPS currently recognizes the following classes:

- "file" − a disk file.
- "direct" − a parallel or fixed−rate serial data port, currently used for Centronics, IEEE−1284, and USB printer ports.
- "serial" − a variable−rate serial port.
- "network" − a network connection, typically via AppSocket, HTTP, IPP, LPD, or SMB/CIFS protocols.

After the device class is the *device URI*, in this case "serial:/dev/ttyS0?baud=115200". This is the URI that should be used by the user to select this port. For serial ports, the "baud=115200" specifies the maximum baud rate supported by the port − the actual value will vary based on the speed the user selects for the printer.

The last two strings are the model and description for the port. The "Unknown" string means that the printer model is unknown − some devices are able to provide a make and model such as "HP DeskJet" that allows users and software to choose an appropriate printer driver more easily. Both the model and description must be enclosed inside double quotes.

Opening the Serial Port

As noted previously, all backends should open device files in exclusive mode, and retry as needed until the port is available. The serial port does this using a `do-while` loop:

```
do
{
  if ((fd = open(resource, O_WRONLY | O_NOCTTY | O_EXCL)) == -1)
  {
    if (errno == EBUSY)
    {
      fputs("INFO: Serial port busy; will retry in 30 seconds...\n", stderr);
      sleep(30);
    }
    else
    {
      perror("ERROR: Unable to open serial port device file");
      return (1);
    }
  }
}
while (fd < 0);
```

If the port is busy or in use by another process, the backend will go to sleep for 30 seconds and try again. If another error is detected a message is sent to the user and the backend aborts the print job until the problem can be corrected.

Writing Data to the Port

Network and character devices pose an interesting problem when writing data to the port − they may not be able to write all of the bytes in your buffer before returning. To work around this problem you must loop until all bytes have been written:

```
while (nbytes > 0)
{
  if ((wbytes = write(fd, bufptr, nbytes)) < 0)
    if (errno == ENOTTY)
      wbytes = write(fd, bufptr, nbytes);

  if (wbytes < 0)
  {
    perror("ERROR: Unable to send print file to printer");
    break;
  }

  nbytes -= wbytes;
  bufptr += wbytes;
```

```
    }
```

The check for the `ENOTTY` error is needed on some platforms to clear an error from a previous `ioctl()` call.

Finishing Up

Once you have sent the print file, return 0 if the file printed successfully or 1 if it did not. This will allow the scheduler to stop the print job if there is a device error, preserving the print job for later printing once the problem has been corrected.

A − Software License Agreement

Common UNIX Printing System License Agreement

44141 AIRPORT VIEW DR STE 204
HOLLYWOOD, MARYLAND 20636−3111 USA

Voice: +1.301.373.9600
Email: cups−info@cups.org
WWW: http://www.cups.org

Introduction

The Common UNIX Printing System™, ("CUPS™"), is provided under the GNU General Public License ("GPL") and GNU Library General Public License ("LGPL"), Version 2. A copy of these licenses follow this introduction.

The GNU LGPL applies to the CUPS API library, located in the "cups" subdirectory of the CUPS source distribution and in the "/usr/include/cups" directory and "libcups.a", "libcups.sl", or "libcups.so" files in the binary distributions.

The GNU GPL applies to the remainder of the CUPS distribution, including the "pstoraster" filter which is based upon GNU Ghostscript 5.50 and the "pdftops" filter which is based upon Xpdf 0.90.

For those not familiar with the GNU GPL, the license basically allows you to:

- Use the CUPS software at no charge.
- Distribute verbatim copies of the software in source or binary form.
- Sell verbatim copies of the software for a media fee, or sell support for the software.
- Distribute or sell printer drivers and filters that use CUPS so long as source code is made available under the GPL.

What this license **does not** allow you to do is make changes or add features to CUPS and then sell a binary distribution without source code. You must provide source for any new drivers, changes, or additions to the software, and all code must be provided under the GPL or LGPL as appropriate.

The GNU LGPL relaxes the "link−to" restriction, allowing you to develop applications that use the CUPS API library under other licenses and/or conditions as appropriate for your application.

Trademarks

Easy Software Products has trademarked the Common UNIX Printing System, CUPS, and CUPS logo. These names and logos may be used freely in any direct port or binary distribution of CUPS. To use them in derivative products, please contract Easy Software Products for written permission. Our intention is to protect the value of these trademarks and ensure that any derivative product meets the same high−quality standards as the original.

Binary Distribution Rights

Easy Software Products also sells rights to the CUPS source code under a binary distribution license for vendors that are unable to release source code for their drivers, additions, and modifications to CUPS under the GNU GPL and LGPL. For information please contact us at the address shown above.

The Common UNIX Printing System provides a "pstoraster" filter that utilizes the GNU GhostScript 5.50 core to convert PostScript files into a stream of raster images. For binary distribution licensing of this software, please contact:

Miles Jones
Director of Marketing
Artifex Software Inc.
454 Las Gallinas Ave., Suite 108
San Rafael, CA 94903 USA
Voice: +1.415.492.9861
Fax: +1.415.492.9862
EMail: info@arsoft.com

The "pdftops" filter is based on the Xpdf 0.90 software. For binary distribution licensing of this software, please contact:

Derek B. Noonburg
Email: derekn@foolabs.com
WWW: http://www.foolabs.com/xpdf/

Support

Easy Software Products sells software support for CUPS as well as a commercial printing product based on CUPS called ESP Print Pro. You can find out more at our web site:

http://www.easysw.com

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software−−to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   c. if the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective

works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
    a. Accompany it with the complete corresponding machine−readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
    b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine−readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
    c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty−free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

GNU LIBRARY GENERAL PUBLIC LICENSE

Version 2, June 1991

[This is the first released version of the library GPL. It is
 numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software−−to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License,

which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non−free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non−free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the

Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a. The modified work must itself be a software library.

b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well−defined independent of the application. Therefore, Subsection 2d requires that any application−supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than

version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine−readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a. Accompany the work with the complete corresponding machine−readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine−readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side−by−side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty−free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

B − Constants

This appendix lists all of the constants that are defined by the CUPS API.

CUPS Constants

Version Number

The `CUPS_VERSION` constant is a floating−point number representing the API version number. The current version number is 1.0100 which represents CUPS version 1.1.0.

Printer Capabilities

The `CUPS_PRINTER` constants represent capability bits for printers and classes:

- `CUPS_PRINTER_LOCAL` − Is a local printer or class.
- `CUPS_PRINTER_REMOTE` − Is a remote printer or class.
- `CUPS_PRINTER_CLASS` − Is a class.
- `CUPS_PRINTER_BW` − Printer prints in black and white.
- `CUPS_PRINTER_COLOR` − Printer prints in color.
- `CUPS_PRINTER_DUPLEX` − Printer can print double−sided.
- `CUPS_PRINTER_STAPLE` − Printer can staple output.
- `CUPS_PRINTER_COPIES` − Printer can produce multiple copies on its own.
- `CUPS_PRINTER_COLLATE` − Printer can collate copies.
- `CUPS_PRINTER_PUNCH` − Printer can punch holes in output.

- CUPS_PRINTER_COVER − Printer can put covers on output.
- CUPS_PRINTER_BIND − Printer can bind output.
- CUPS_PRINTER_SORT − Printer can sort output.
- CUPS_PRINTER_SMALL − Printer can print on media up to 9x14 inches.
- CUPS_PRINTER_MEDIUM − Printer can print on media from 9x14 to 18x24 inches.
- CUPS_PRINTER_LARGE − Printer can print on media larger than 18x24 inches.
- CUPS_PRINTER_VARIABLE − Printer can print on variable or custom media sizes.
- CUPS_PRINTER_IMPLICIT − Is an implicit class.
- CUPS_PRINTER_OPTIONS − All of the printer capability and option bits.

Encodings

CUPS defines the following character set encoding constants:

- CUPS_US_ASCII − US ASCII character set.
- CUPS_UTF_8 − UTF−8 encoding of Unicode.
- CUPS_ISO8859_1 − ISO−8859−1 character set.
- CUPS_ISO8859_2 − ISO−8859−2 character set.
- CUPS_ISO8859_3 − ISO−8859−3 character set.
- CUPS_ISO8859_4 − ISO−8859−4 character set.
- CUPS_ISO8859_5 − ISO−8859−5 character set.
- CUPS_ISO8859_6 − ISO−8859−6 character set.
- CUPS_ISO8859_7 − ISO−8859−7 character set.
- CUPS_ISO8859_8 − ISO−8859−8 character set.
- CUPS_ISO8859_9 − ISO−8859−9 character set.
- CUPS_ISO8859_10 − ISO−8859−10 character set.
- CUPS_ISO8859_13 − ISO−8859−13 character set.
- CUPS_ISO8859_14 − ISO−8859−14 character set.
- CUPS_ISO8859_15 − ISO−8859−15 character set.
- CUPS_WINDOWS_874 − Windows code page 874.
- CUPS_WINDOWS_1250 − Windows code page 1250.
- CUPS_WINDOWS_1251 − Windows code page 1251.
- CUPS_WINDOWS_1252 − Windows code page 1252.
- CUPS_WINDOWS_1253 − Windows code page 1253.
- CUPS_WINDOWS_1254 − Windows code page 1254.
- CUPS_WINDOWS_1255 − Windows code page 1255.
- CUPS_WINDOWS_1256 − Windows code page 1256.
- CUPS_WINDOWS_1257 − Windows code page 1257.
- CUPS_WINDOWS_1258 − Windows code page 1258.

HTTP Constants

Limits

The following constants define the limits for strings:

- HTTP_MAX_BUFFER − Size of socket buffer.
- HTTP_MAX_HOST − Maximum length of hostname.
- HTTP_MAX_URI − Maximum length of URI.
- HTTP_MAX_VALUE − Maximum length of field values.

Status Codes

The following status codes can be returned by `httpUpdate()`:

- `HTTP_ERROR` − A network error occurred
- `HTTP_CONTINUE` − Continue response from HTTP proxy
- `HTTP_OK` − OPTIONS/GET/HEAD/POST/TRACE command was successful
- `HTTP_CREATED` − PUT command was successful
- `HTTP_ACCEPTED` − DELETE command was successful
- `HTTP_NOT_AUTHORITATIVE` − Information isn't authoritative
- `HTTP_NO_CONTENT` − Successful command
- `HTTP_RESET_CONTENT` − Content was reset/recreated
- `HTTP_PARTIAL_CONTENT` − Only a partial file was recieved/sent
- `HTTP_MULTIPLE_CHOICES` − Multiple files match request
- `HTTP_MOVED_PERMANENTLY` − Document has moved permanently
- `HTTP_MOVED_TEMPORARILY` − Document has moved temporarily
- `HTTP_SEE_OTHER` − See this other link...
- `HTTP_NOT_MODIFIED` − File not modified
- `HTTP_USE_PROXY` − Must use a proxy to access this URI
- `HTTP_BAD_REQUEST` − Bad request
- `HTTP_UNAUTHORIZED` − Unauthorized to access host
- `HTTP_PAYMENT_REQUIRED` − Payment required
- `HTTP_FORBIDDEN` − Forbidden to access this URI
- `HTTP_NOT_FOUND` − URI was not found
- `HTTP_METHOD_NOT_ALLOWED` − Method is not allowed
- `HTTP_NOT_ACCEPTABLE` − Not Acceptable
- `HTTP_PROXY_AUTHENTICATION` − Proxy Authentication is Required
- `HTTP_REQUEST_TIMEOUT` − Request timed out
- `HTTP_CONFLICT` − Request is self−conflicting
- `HTTP_GONE` − Server has gone away
- `HTTP_LENGTH_REQUIRED` − A content length or encoding is required
- `HTTP_PRECONDITION` − Precondition failed
- `HTTP_REQUEST_TOO_LARGE` − Request entity too large
- `HTTP_URI_TOO_LONG` − URI too long
- `HTTP_UNSUPPORTED_MEDIATYPE` − The requested media type is unsupported
- `HTTP_SERVER_ERROR` − Internal server error
- `HTTP_NOT_IMPLEMENTED` − Feature not implemented
- `HTTP_BAD_GATEWAY` − Bad gateway
- `HTTP_SERVICE_UNAVAILABLE` − Service is unavailable
- `HTTP_GATEWAY_TIMEOUT` − Gateway connection timed out
- `HTTP_NOT_SUPPORTED` − HTTP version not supported

Fields

The following fields are indices for each of the standard HTTP fields in HTTP 1/1:

- `HTTP_FIELD_ACCEPT_LANGUAGE` − Accept−Language
- `HTTP_FIELD_ACCEPT_RANGES` − Accept−Ranges
- `HTTP_FIELD_AUTHORIZATION` − Authorization
- `HTTP_FIELD_CONNECTION` − Connection

- HTTP_FIELD_CONTENT_ENCODING − Content−Encoding
- HTTP_FIELD_CONTENT_LANGUAGE − Content−Language
- HTTP_FIELD_CONTENT_LENGTH − Content−Length
- HTTP_FIELD_CONTENT_LOCATION − Content−Location
- HTTP_FIELD_CONTENT_MD5 − Content−MD5
- HTTP_FIELD_CONTENT_RANGE − Content−Range
- HTTP_FIELD_CONTENT_TYPE − Content−Type
- HTTP_FIELD_CONTENT_VERSION − Content−Version
- HTTP_FIELD_DATE − Date
- HTTP_FIELD_HOST − Host
- HTTP_FIELD_IF_MODIFIED_SINCE − If−Modified−Since
- HTTP_FIELD_IF_UNMODIFIED_SINCE − If−Unmodified−Since
- HTTP_FIELD_KEEP_ALIVE − Keep−Alive
- HTTP_FIELD_LAST_MODIFIED − Last−Modified
- HTTP_FIELD_LINK − Link
- HTTP_FIELD_LOCATION − Location
- HTTP_FIELD_RANGE − Range
- HTTP_FIELD_REFERER − Referer
- HTTP_FIELD_RETRY_AFTER − Retry−After
- HTTP_FIELD_TRANSFER_ENCODING − Transfer−Encoding
- HTTP_FIELD_UPGRADE − Upgrade
- HTTP_FIELD_USER_AGENT − User−Agent
- HTTP_FIELD_WWW_AUTHENTICATE − WWW−Authenticate

IPP Constants

Limits

The following constants define array limits for IPP data:

- IPP_MAX_NAME − Maximum length of an attribute name
- IPP_MAX_VALUES − Maximum number of set−of values that can be read in a request.

Tags

- IPP_TAG_ZERO − Wildcard tag value for searches; also used to separate groups of attributes
- IPP_TAG_OPERATION − Tag for values of type operation
- IPP_TAG_JOB − Tag for values of type job
- IPP_TAG_END − Tag for values of type end
- IPP_TAG_PRINTER − Tag for values of type printer
- IPP_TAG_UNSUPPORTED_GROUP − Tag for values of type unsupported_group
- IPP_TAG_UNSUPPORTED_VALUE − Tag for values of type unsupported_value
- IPP_TAG_DEFAULT − Tag for values of type default
- IPP_TAG_UNKNOWN − Tag for values of type unknown
- IPP_TAG_NOVALUE − Tag for values of type novalue
- IPP_TAG_NOTSETTABLE − Tag for values of type notsettable
- IPP_TAG_DELETEATTR − Tag for values of type deleteattr
- IPP_TAG_ANYVALUE − Tag for values of type anyvalue
- IPP_TAG_INTEGER − Tag for values of type integer
- IPP_TAG_BOOLEAN − Tag for values of type boolean

- IPP_TAG_ENUM − Tag for values of type enum
- IPP_TAG_STRING − Tag for values of type string
- IPP_TAG_DATE − Tag for values of type date
- IPP_TAG_RESOLUTION − Tag for values of type resolution
- IPP_TAG_RANGE − Tag for values of type range
- IPP_TAG_COLLECTION − Tag for values of type collection
- IPP_TAG_TEXTLANG − Tag for values of type textlang
- IPP_TAG_NAMELANG − Tag for values of type namelang
- IPP_TAG_TEXT − Tag for values of type text
- IPP_TAG_NAME − Tag for values of type name
- IPP_TAG_KEYWORD − Tag for values of type keyword
- IPP_TAG_URI − Tag for values of type uri
- IPP_TAG_URISCHEME − Tag for values of type urischeme
- IPP_TAG_CHARSET − Tag for values of type charset
- IPP_TAG_LANGUAGE − Tag for values of type language
- IPP_TAG_MIMETYPE − Tag for values of type mimetype

Resolution Units

The IPP_RES_PER_INCH and IPP_RES_PER_CM constants specify dots per inch and dots per centimeter, respectively.

Finishings

The finishing values specify special finishing operations to be performed on the job.

- IPP_FINISH_NONE − Do no finishing
- IPP_FINISH_STAPLE − Staple the job
- IPP_FINISH_PUNCH − Punch the job
- IPP_FINISH_COVER − Cover the job
- IPP_FINISH_BIND − Bind the job

Orientations

The orientation values specify the orientation of the job.

- IPP_PORTRAIT − No rotation
- IPP_LANDSCAPE − 90 degrees counter−clockwise
- IPP_REVERSE_LANDSCAPE − 90 degrees clockwise
- IPP_REVERSE_PORTRAIT − 180 degrees

Qualities

The quality values specify the desired quality of the print.

- IPP_QUALITY_DRAFT − Draft quality
- IPP_QUALITY_NORMAL − Normal quality
- IPP_QUALITY_HIGH − High quality

Job States

The job state values are used to represent the current job state.

- `IPP_JOB_PENDING` − Job is pending
- `IPP_JOB_HELD` − Job is held
- `IPP_JOB_PROCESSING` − Job is processing
- `IPP_JOB_STOPPED` − Job is stopped
- `IPP_JOB_CANCELLED` − Job is cancelled
- `IPP_JOB_ABORTED` − Job is aborted
- `IPP_JOB_COMPLETED` − Job is completed

Printer States

The printer state values are used to represent the current printer state.

- `IPP_PRINTER_IDLE` − Printer is idle
- `IPP_PRINTER_PROCESSING` − Printer is processing
- `IPP_PRINTER_STOPPED` − Printer is stopped

Operations

The operation values represent the available IPP operations.

- `IPP_PRINT_JOB` − Print a file
- `IPP_PRINT_URI` − Print a URI
- `IPP_VALIDATE_JOB` − Validate job attributes
- `IPP_CREATE_JOB` − Create a new job
- `IPP_SEND_DOCUMENT` − Send a document to a job
- `IPP_SEND_URI` − Send a URI to a job
- `IPP_CANCEL_JOB` − Cancel a job
- `IPP_GET_JOB_ATTRIBUTES` − Get job attributes
- `IPP_GET_JOBS` − Get a list of all jobs
- `IPP_GET_PRINTER_ATTRIBUTES` − Get printer attributes
- `IPP_HOLD_JOB` − Hold a pending job
- `IPP_RELEASE_JOB` − Release a held job
- `IPP_RESTART_JOB` − Restart a completed job
- `IPP_PAUSE_PRINTER` − Pause a printer
- `IPP_RESUME_PRINTER` − Restart a paused printer
- `IPP_PURGE_JOBS` − Purge jobs from the queue
- `IPP_SET_PRINTER_ATTRIBUTES` − Set printer attributes
- `IPP_SET_JOB_ATTRIBUTES` − Set job attributes
- `IPP_GET_PRINTER_SUPPORTED_VALUES` − Get printer supported values
- `CUPS_GET_DEFAULT` − Get the default destination
- `CUPS_GET_PRINTERS` − Get a list of all printers
- `CUPS_ADD_PRINTER` − Add or modify a printer
- `CUPS_DELETE_PRINTER` − Delete a printer
- `CUPS_GET_CLASSES` − Get a list of all classes
- `CUPS_ADD_CLASS` − Add or modify a class
- `CUPS_DELETE_CLASS` − Delete a class

- CUPS_ACCEPT_JOBS − Accept jobs on a printer or class
- CUPS_REJECT_JOBS − Reject jobs on a printer or class
- CUPS_SET_DEFAULT − Set the default destination
- CUPS_GET_DEVICES − Get a list of all devices
- CUPS_GET_PPDS − Get a list of all PPDs
- CUPS_MOVE_JOB − Move a job to a new destination

Status Codes

Status codes are returned by all IPP requests.

- IPP_OK − Request completed with no errors
- IPP_OK_SUBST − Request completed but some attribute values were substituted
- IPP_OK_CONFLICT − Request completed but some attributes conflicted
- IPP_BAD_REQUEST − The request was bad
- IPP_FORBIDDEN − You don't have access to the resource
- IPP_NOT_AUTHENTICATED − You are not authenticated for the resource
- IPP_NOT_AUTHORIZED − You not authorized to access the resource
- IPP_NOT_POSSIBLE − The requested operation cannot be completed
- IPP_TIMEOUT − A timeout occurred
- IPP_NOT_FOUND − The resource was not found
- IPP_GONE − The resource has gone away
- IPP_REQUEST_ENTITY − The request was too large
- IPP_REQUEST_VALUE − The request contained a value that was unknown to the server
- IPP_DOCUMENT_FORMAT − The document format is not supported by the server
- IPP_ATTRIBUTES − Required attributes are missing
- IPP_URI_SCHEME − The URI scheme is not supported
- IPP_CHARSET − The charset is not supported
- IPP_CONFLICT − One or more attributes conflict
- IPP_COMPRESSION_NOT_SUPPORTED − The specified compression is not supported
- IPP_COMPRESSION_ERROR − The compressed data contained an error
- IPP_DOCUMENT_FORMAT_ERROR − The document data contained an error in it
- IPP_DOCUMENT_ACCESS_ERROR − The remote document could not be accessed
- IPP_INTERNAL_ERROR − The server encountered an internal error
- IPP_OPERATION_NOT_SUPPORTED − The requested operation is not supported
- IPP_SERVICE_UNAVAILABLE − The requested service is unavailable
- IPP_VERSION_NOT_SUPPORTED − The IPP request version is not supported
- IPP_DEVICE_ERROR − The output device encountered an error
- IPP_TEMPORARY_ERROR − A temporary error occurred
- IPP_NOT_ACCEPTING − The destination is not accepting jobs
- IPP_PRINTER_BUSY − The destination is busy
- IPP_ERROR_JOB_CANCELLED − The requested job has been cancelled
- IPP_MULTIPLE_JOBS_NOT_SUPPORTED − The server does not support multiple jobs

PPD Constants

PPD Format Version

The PPD_VERSION constant defines a floating point number representing the newest format version that is supported by CUPS, currently 4.3.

PPD User−Interface Types

Each printer option has a type associated with it:

- `PPD_UI_BOOLEAN` − The user can turn this option on or off
- `PPD_UI_PICKONE` − The user can choose one option value to use.
- `PPD_UI_PICKMANY` − The user can choose zero or more option values.

PPD Sections

Some options must be output before others, or in different sections of the output document. The `ppd_section_t` enumeration defines which section the option must be output in:

- `PPD_ORDER_ANY` − The option can be output in any of the document, page, or prolog sections of the document
- `PPD_ORDER_DOCUMENT` − The option must be output in the DocumentSetup section of the document
- `PPD_ORDER_EXIT` − The option must be output before the document
- `PPD_ORDER_JCL` − The option must be output in the job control section of the document
- `PPD_ORDER_PAGE` − The option must be output in the PageSetup section of the document
- `PPD_ORDER_PROLOG` − The option must be output in the Prolog section of the document

PPD Colorspaces

Each printer has a default colorspace:

- `PPD_CS_CMYK` − The printer uses CMYK colors by default
- `PPD_CS_CMY` − The printer uses CMY colors by default
- `PPD_CS_GRAY` − The printer uses grayscale by default
- `PPD_CS_RGB` − The printer uses RGB colors by default
- `PPD_CS_RGBK` − The printer uses RGBK colors by default
- `PPD_CS_N` − The printer uses a DeviceN colorspace by default

Raster Constants

Raster Sync Words

The `CUPS_RASTER_SYNC` and `CUPS_RASTER_REVSYNC` constants define the standard sync words at the beginning of each CUPS raster file.

Raster Stream Modes

The `CUPS_RASTER_READ` and `CUPS_RASTER_WRITE` constants are used with the `cupsRasterOpen()` function to specify a stream for reading or writing.

Raster Boolean Constants

The `CUPS_FALSE` and `CUPS_TRUE` constants represent boolean values in the page header.

Raster Jog Values

The `cups_jog_t` enumeration defines constants for the Jog page device dictionary variable:

- `CUPS_JOG_NONE` − Do no jogging
- `CUPS_JOG_FILE` − Jog pages after each file
- `CUPS_JOG_JOB` − Jog pages after each job
- `CUPS_JOG_SET` − Jog pages after each set of jobs

Raster Orientation Values

The `cups_orient_t` enumeration defines constants for the Orientation page device dictionary variable:

- `CUPS_ORIENT_0` − Portrait orientation
- `CUPS_ORIENT_90` − Landscape orientation
- `CUPS_ORIENT_180` − Reverse−portrait orientation
- `CUPS_ORIENT_270` − Reverse−landscape orientation

Raster CutMedia Values

The `cups_cut_t` enumeration defines constants for the CutMedia page device dictionary variable:

- `CUPS_CUT_NONE` − Do no jogging
- `CUPS_CUT_FILE` − Cut pages after each file
- `CUPS_CUT_JOB` − Cut pages after each job
- `CUPS_CUT_SET` − Cut pages after each set of jobs
- `CUPS_CUT_PAGE` − Cut each page

Raster AdvanceMedia Values

The `cups_advance_t` enumeration defines constants for the AdvanceMedia page device dictionary variable:

- `CUPS_ADVANCE_NONE` − Do no jogging
- `CUPS_ADVANCE_FILE` − Advance media after each file
- `CUPS_ADVANCE_JOB` − Advance media after each job
- `CUPS_ADVANCE_SET` − Advance media after each set of jobs
- `CUPS_ADVANCE_PAGE` − Advance media for each page

Raster LeadingEdge Values

The `cups_edge_t` enumeration defines constants for the LeadingEdge page device dictionary variable:

- `CUPS_EDGE_TOP` − The top of the media is the leading edge
- `CUPS_EDGE_RIGHT` − The right of the media is the leading edge
- `CUPS_EDGE_BOTTOM` − The bottom of the media is the leading edge
- `CUPS_EDGE_LEFT` − The left of the media is the leading edge

Raster Color Order Values

The `cups_order_t` enumeration defines the possible color value orderings:

- `CUPS_ORDER_CHUNKED` − CMYK CMYK CMYK
- `CUPS_ORDER_BANDED` − CCC MMM YYY KKK
- `CUPS_ORDER_PLANAR` − CCC ... MMM ... YYY ... KKK ...

Raster Colorspace Values

The `cups_cspace_t` enumeration defines the possible colorspaces:

- `CUPS_CSPACE_W` − White (luminance)
- `CUPS_CSPACE_RGB` − Red, green, blue
- `CUPS_CSPACE_RGBA` − Red, green, blue, alpha
- `CUPS_CSPACE_K` − Black
- `CUPS_CSPACE_CMY` − Cyan, magenta, yellow
- `CUPS_CSPACE_YMC` − Yellow, magenta, cyan
- `CUPS_CSPACE_CMYK` − Cyan, magenta, yellow, black
- `CUPS_CSPACE_YMCK` − Yellow, magenta, cyan, black
- `CUPS_CSPACE_KCMY` − Black, cyan, magenta, yellow
- `CUPS_CSPACE_KCMYcm` − Black, cyan, magenta, yellow, light cyan, light magenta
- `CUPS_CSPACE_GMCK` − Metallic yellow (gold), metallic magenta, metallic cyan, black
- `CUPS_CSPACE_GMCS` − Metallic yellow (gold), metallic magenta, metallic cyan, metallic grey (silver)
- `CUPS_CSPACE_WHITE` − White pigment (black as white pigment)
- `CUPS_CSPACE_GOLD` − Gold foil (black as gold foil)
- `CUPS_CSPACE_SILVER` − Silver foil (black as silver foil)

C − Structures

This appendix describes all of the structures that are defined by the CUPS API.

Raster Structures

Raster Page Header

The raster page header consists of the PostScript page device dictionary for the page:

Member	Type	Description
MediaClass	char[64]	The media class name
MediaColor	char[64]	The media color name
MediaType	char[64]	The media type name
OutputType	char[64]	The output type name
AdvanceDistance	unsigned	The distance to advance the media in points
AdvanceMedia	cups_adv_t	When to advance the media
Collate	cups_bool_t	Whether or not to produce collated copies
CutMedia	cups_cut_t	When to cut the media

Duplex	cups_bool_t	Whether or not to print on both sides of the paper
HWResolution	unsigned[2]	The resolution of the page image in pixels per inch; the HWResolution[0] represents the horizontal resolution and HWResolution[1] represents the vertical resolution
ImagingBoundingBox	unsigned[4]	The bounding box for the page in points; the elements represent the left, bottom, right, and top coordinates of the imaged area (if 0 then the whole page is imaged)
InsertSheet	cups_bool_t	Whether or not to insert a sheet before this page
Jog	cups_jog_t	When to jog copies of the page
LeadingEdge	cups_edge_t	The leading edge of the page
Margins	unsigned[2]	The lower−lefthand margin of the page in points
ManualFeed	cups_bool_t	Whether or not to manually feed the page
MediaPosition	unsigned	The input slot number to use
MediaWeight	unsigned	The weight of the output media in grams/m^2
MirrorPrint	cups_bool_t	Whether or not to mirror the print
NegativePrint	cups_bool_t	Whether or not to invert the print
NumCopies	unsigned	The number of copies to produce
Orientation	cups_orient_t	The orientation of the page image
OutputFaceUp	cups_bool_t	Whether or not to output the page face up
PageSize	unsigned[2]	The width and height of the page in points
Separations	cups_bool_t	Whether or not to output separations
TraySwitch	cups_bool_t	Whether or not to automatically switch trays for the requested media size/type

Tumble	cups_bool_t	Whether or not to rotate the back side of the page
cupsWidth	unsigned	The width of the page image in pixels
cupsHeight	unsigned	The height of the page image in pixels
cupsMediaType	unsigned	The device−specific media type code
cupsBitsPerColor	unsigned	The number of bits per color
cupsBitsPerPixel	unsigned	The number of bits per pixel
cupsBytesPerLine	unsigned	The number of bytes per line of image data
cupsColorOrder	cups_order_t	The order of color values
cupsColorSpace	cups_cspace_t	The type of color values
cupsCompression	unsigned	The device−specific compression code
cupsRowCount	unsigned	The device−specific row count
cupsRowFeed	unsigned	The device−specific row feed
cupsRowStep	unsigned	The device−specific row step

D − Functions

This appendix provides a reference for all of the CUPS API functions.

cupsAddOption()

Usage

```
int
cupsAddOption(const char *name,
              const char *value,
              int num_options,
              cups_option_t **options);
```

Arguments

Argument	Description
name	The name of the option.
value	The value of the option.
num_options	Number of options currently in the array.
options	Pointer to the options array.

Returns

The new number of options.

Description

`cupsAddOption()` adds an option to the specified array.

Example

```
#include <cups.h>

...

/* Declare the options array */
int           num_options;
cups_option_t *options;

/* Initialize the options array */
num_options = 0;
options     = (cups_option_t *)0;

/* Add options using cupsAddOption() */
num_options = cupsAddOption("media", "letter", num_options, &options);
num_options = cupsAddOption("resolution", "300dpi", num_options, &options);
```

See Also

`cupsFreeOptions()`, `cupsGetOption()`, `cupsParseOptions()`

cupsCancelJob()

Usage

```
int
cupsCancelJob(const char *dest,
              int job);
```

Arguments

Argument	Description
dest	Printer or class name
job	Job ID

Returns

1 on success, 0 on failure. On failure the error can be found by calling `cupsLastError()`.

Description

`cupsCancelJob()` cancels the specifies job.

Example

```
#include <cups.h>

cupsCancelJob("LaserJet", 1);
```

See Also

`cupsLastError()`, `cupsPrintFile()`

cupsDoFileRequest()

Usage

```
ipp_t *
cupsDoFileRequest(http_t *http,
                  ipp_t *request,
                  const char *resource,
                  const char *filename);
```

Arguments

Argument	Description
http	HTTP connection to server.
request	IPP request data.
resource	HTTP resource name for POST.
filename	File to send with POST request (`NULL` pointer if none.)

Returns

IPP response data or `NULL` if the request fails. On failure the error can be found by calling `cupsLastError()`.

Description

`cupsDoFileRequest()` does a HTTP POST request and provides the IPP request and optionally the contents of a file to the IPP server. It also handles resubmitting the request and performing password authentication as needed.

Example

```
#include <cups.h>

http_t      *http;
cups_lang_t *language;
ipp_t       *request;
ipp_t       *response;

...

/* Get the default language */
language = cupsLangDefault();

/* Create a new IPP request */
request  = ippNew();

request->request.op.operation_id = IPP_PRINT_FILE;
request->request.op.request_id   = 1;

/* Add required attributes */
```

```
ippAddString(request, IPP_TAG_OPERATION, IPP_TAG_CHARSET,
             "attributes-charset", NULL, cupsLangEncoding(language));

ippAddString(request, IPP_TAG_OPERATION, IPP_TAG_LANGUAGE,
             "attributes-natural-language", NULL,
             language != NULL ? language->language : "C");

ippAddString(request, IPP_TAG_OPERATION, IPP_TAG_URI, "printer-uri",
             NULL, "ipp://hostname/resource");

ippAddString(request, IPP_TAG_OPERATION, IPP_TAG_NAME, "requesting-user-name",
             NULL, cupsUser());

/* Do the request... */
response = cupsDoFileRequest(http, request, "/resource", "filename.txt");
```

See Also

cupsLangDefault(), cupsLangEncoding(), cupsUser(), httpConnect(), ippAddString(), ippNew()

cupsDoRequest()

Usage

```
ipp_t *
cupsDoRequest(http_t *http,
              ipp_t *request,
              const char *resource);
```

Arguments

Argument	Description
http	HTTP connection to server.
request	IPP request data.
resource	HTTP resource name for POST.

Returns

IPP response data or `NULL` if the request fails. On failure the error can be found by calling `cupsLastError()`.

Description

`cupsDoRequest()` does a HTTP POST request and provides the IPP request to the IPP server. It also handles resubmitting the request and performing password authentication as needed.

Example

```
#include <cups.h>

http_t      *http;
cups_lang_t *language;
ipp_t       *request;
ipp_t       *response;

...

/* Get the default language */
language = cupsLangDefault();

/* Create a new IPP request */
request  = ippNew();

request->request.op.operation_id = IPP_GET_PRINTER_ATTRIBUTES;
request->request.op.request_id   = 1;

/* Add required attributes */
ippAddString(request, IPP_TAG_OPERATION, IPP_TAG_CHARSET,
             "attributes-charset", NULL, cupsLangEncoding(language));

ippAddString(request, IPP_TAG_OPERATION, IPP_TAG_LANGUAGE,
             "attributes-natural-language", NULL,
```

```
              language != NULL ? language->language : "C");

ippAddString(request, IPP_TAG_OPERATION, IPP_TAG_URI, "printer-uri",
              NULL, "ipp://hostname/resource");

/* Do the request... */
response = cupsDoRequest(http, request, "/resource");
```

See Also

cupsLangDefault(), cupsLangEncoding(), cupsUser(), httpConnect(), ippAddString(), ippNew()

cupsFreeOptions()

Usage

```
void
cupsFreeOptions(int num_options,
                cups_option_t *options);
```

Arguments

Argument	Description
num_options	Number of options in array.
options	Pointer to options array.

Description

`cupsFreeOptions()` frees all memory associated with the option array specified.

Example

```
#include <cups/cups.h>

int           num_options;
cups_option_t *options;

...

cupsFreeOptions(num_options, options);
```

See Also

cupsAddOption(), cupsGetOption(), cupsMarkOptions(), cupsParseOptions()

cupsGetClasses()

Usage

```
int
cupsGetClasses(char ***classes);
```

Arguments

Argument	Description
classes	Pointer to character pointer array.

Returns

The number of printer classes available.

Description

`cupsGetClasses()` gets a list of the available printer classes. The returned array should be freed using the `free()` when it is no longer needed.

Example

```
#include <cups/cups.h>

int  i;
int  num_classes;
char **classes;

...

num_classes = cupsGetClasses(

...

if (num_classes > 0)
{
  for (i = 0; i num_classes; i ++)
    free(classes[i]);

  free(classes);
}
```

See Also

cupsGetDefault(), cupsGetPrinters()

cupsGetDefault()

Usage

```
const char *
cupsGetDefault(void);
```

Returns

A pointer to the default destination.

Description

`cupsGetDefault()` gets the default destination printer or class. The default destination is stored in a static string and will be overwritten (usually with the same value) after each call.

Example

```
#include <cups/cups.h>

printf("The default destination is %s\n", cupsGetDefault());
```

See Also

cupsGetClasses(), cupsGetPrinters()

cupsGetOption()

Usage

```
const char *
cupsGetOption(const char *name,
              int num_options,
              cups_option_t *options);
```

Arguments

Argument	Description
name	The name of the option.
num_options	The number of options in the array.
options	The options array.

Returns

A pointer to the option values or `NULL` if the option is not defined.

Description

`cupsGetOption()` returns the first occurrence of the named option. If the option is not included in the options array then a `NULL` pointer is returned.

```
#include <cups/cups.h>

int           num_options;
cups_option_t *options;
const char    *media;

...

media = cupsGetOption("media", num_options, options);
```

See Also

cupsAddOption(), cupsFreeOptions(), cupsMarkOptions(), cupsParseOptions()

cupsGetPassword()

Usage

```
const char *
cupsGetPassword(const char *prompt);
```

Arguments

Argument	Description
prompt	The prompt to display to the user.

Returns

A pointer to the password that was entered or `NULL` if no password was entered.

Description

`cupsGetPassword()` displays the prompt string and asks the user for a password. The password text is not echoed to the user.

Example

```
#include <cups/cups.h>

char *password;

...

password = cupsGetPassword("Please enter a password:");
```

See Also

cupsServer(), cupsSetPasswordCB(), cupsSetServer(), cupsSetUser(), cupsUser()

cupsGetPPD()

Usage

```
const char *
cupsGetPPD(const char *printer);
```

Arguments

Argument	Description
printer	The name of the printer.

Returns

The name of a temporary file containing the PPD file or `NULL` if the printer cannot be located or does not have a PPD file.

Description

`cupsGetPPD()` gets a copy of the PPD file for the named printer. The printer name can be of the form "printer" or "printer@hostname".

You should remove (unlink) the PPD file after you are done using it. The filename is stored in a static buffer and will be overwritten with each call to `cupsGetPPD()`.

Example

```
#include <cups/cups.h>

char *ppd;

...

ppd = cupsGetPPD("printer@hostname");

...

unlink(ppd);
```

cupsGetPrinters()

Usage

```
int
cupsGetPrinters(char ***printers);
```

Arguments

Argument	Description
printers	Pointer to character pointer array.

Returns

The number of printer printers available.

Description

`cupsGetPrinters()` gets a list of the available printers. The returned array should be freed using the `free()` when it is no longer needed.

Example

```
#include <cups/cups.h>

int  i;
int  num_printers;
char **printers;

...

num_printers = cupsGetPrinters(

...

if (num_printers > 0)
{
  for (i = 0; i num_printers; i ++)
    free(printers[i]);

  free(printers);
}
```

See Also

cupsGetClasses(), cupsGetDefault()

cupsLangDefault()

Usage

```
const char *
cupsLangDefault(void);
```

Returns

A pointer to the default language structure.

Description

`cupsLangDefault()` returns a language structure for the default language. The default language is defined by the `LANG` environment variable. If the specified language cannot be located then the POSIX (English) locale is used.

Call `cupsLangFree()` to free any memory associated with the language structure when you are done.

Example

```
#include <cups/language.h>

cups_lang_t *language;
...

language = cupsLangDefault();

...

cupsLangFree(language);
```

See Also

cupsLangEncoding(), cupsLangFlush(), cupsLangFree(), cupsLangGet(), cupsLangString()

cupsLangEncoding()

Usage

```
char *
cupsLangEncoding(cups_lang_t *language);
```

Arguments

Argument	Description
language	The language structure.

Returns

A pointer to the encoding string.

Description

`cupsLangEncoding()` returns the language encoding used for the specified language, e.g. "iso−8859−1", "utf−8", etc.

Example

```
#include <cups/language.h>

cups_lang_t *language;
char        *encoding;
...

language = cupsLangDefault();
encoding = cupsLangEncoding(language);
...

cupsLangFree(language);
```

See Also

cupsLangDefault(), cupsLangFlush(), cupsLangFree(), cupsLangGet(), cupsLangString()

cupsLangFlush()

Usage

```
void
cupsLangFlush(void);
```

Description

`cupsLangFlush()` frees all language structures that have been allocated.

Example

```
#include <cups/language.h>

...

cupsLangFlush();
```

See Also

cupsLangDefault(), cupsLangEncoding(), cupsLangFree(), cupsLangGet(), cupsLangString()

cupsLangFree()

Usage

```
void
cupsLangFree(cups_lang_t *language);
```

Arguments

Argument	Description
language	The language structure to free.

Description

`cupsLangFree()` frees the specified language structure.

Example

```
#include <cups/language.h>

cups_lang_t *language;
...

cupsLangFree(language);
```

See Also

cupsLangDefault(), cupsLangEncoding(), cupsLangFlush(), cupsLangGet(), cupsLangString()

cupsLangGet()

Usage

```
cups_lang_t *
cupsLangGet(const char *name);
```

Arguments

Argument	Description
name	The name of the locale.

Returns

A pointer to a language structure.

Description

`cupsLangGet()` returns a language structure for the specified locale. If the locale is not defined then the POSIX (English) locale is substituted.

Example

```
#include <cups/language.h>

cups_lang_t *language;

...

language = cupsLangGet("fr");

...

cupsLangFree(language);
```

See Also

cupsLangDefault(), cupsLangEncoding(), cupsLangFlush(), cupsLangFree(), cupsLangString()

cupsLangString()

Usage

```
char *
cupsLangString(cups_lang_t *language,
               int         message);
```

Arguments

Argument	Description
language	The language to query.
message	The message number.

Returns

A pointer to the message string or `NULL` if the message is not defined.

Description

`cupsLangString()` returns a pointer to the specified message string in the specified language.

Example

```
#include <cups/language.h>

cups_lang_t *language;
char        *s;
...

language = cupsLangGet("fr");

s = cupsLangString(language, CUPS_MSG_YES);

...

cupsLangFree(language);
```

See Also

cupsLangDefault(), cupsLangEncoding(), cupsLangFlush(), cupsLangFree(), cupsLangGet()

cupsLastError()

Usage

```
ipp_status_t
cupsLastError(void);
```

Returns

An enumeration containing the last IPP error.

Description

`cupsLastError()` returns the last IPP error that occurred. If no error occurred then it will return `IPP_OK` or `IPP_OK_CONFLICT`.

Example

```
#include <cups/cups.h>

ipp_status_t status;

...

status = cupsLastError();
```

See Also

cupsCancelJob(), cupsPrintFile()

cupsMarkOptions()

Usage

```
int
cupsMarkOptions(ppd_file_t *ppd,
                int num_options,
                cups_option_t *options);
```

Arguments

Argument	Description
ppd	The PPD file to mark.
num_options	The number of options in the options array.
options	A pointer to the options array.

Returns

The number of conflicts found.

Description

`cupsMarkOptions()` marks options in the PPD file. It also handles mapping of IPP option names and values to PPD option names.

Example

```
#include <cups/cups.h>

int           num_options;
cups_option_t *options;
ppd_file_t    *ppd;

...

cupsMarkOptions(ppd, num_options, options);
```

See Also

cupsAddOption(), cupsFreeOptions(), cupsGetOption(), cupsParseOptions()

cupsParseOptions()

Usage

```
int
cupsParseOptions(const char *arg,
                 int num_options,
                 cups_option_t **options);
```

Arguments

Argument	Description
arg	The string containing one or more options.
num_options	The number of options in the options array.
options	A pointer to the options array pointer.

Returns

The new number of options in the array.

Description

`cupsParseOptions()` parses the specifies string for one or more options of the form "name=value", "name", or "noname". It can be called multiple times to combine the options from several strings.

Example

```
#include <cups/cups.h>

int           num_options;
cups_option_t *options;

...

num_options = 0;
options     = (cups_option_t *)0;
num_options = cupsParseOptions(argv[5], num_options, &options);
```

See Also

cupsAddOption(), cupsFreeOptions(), cupsGetOption(), cupsMarkOptions()

cupsPrintFile()

Usage

```
int
cupsPrintFile(const char    *printer,
              const char    *filename,
              const char    *title,
              int           num_options,
              cups_option_t *options);
```

Arguments

Argument	Description
printer	The printer or class to print to.
filename	The file to print.
title	The job title.
num_options	The number of options in the options array.
options	A pointer to the options array.

Returns

The new job ID number or 0 on error.

Description

`cupsPrintFile()` sends a file to the specified printer or class for printing. If the job cannot be printed the error code can be found by calling `cupsLastError()`.

Example

```
#include <cups/cups.h>

int           num_options;
cups_option_t *options;
int           jobid;

...

jobid = cupsPrintFile("printer@hostname", "filename.ps", "Job Title",
                      num_options, options);
```

See Also

cupsCancelJob(), cupsLastError(), cupsPrintFiles()

cupsPrintFiles()

Usage

```
int
cupsPrintFiles(const char    *printer,
               int           num_files,
               const char    **files,
               const char    *title,
               int           num_options,
               cups_option_t *options);
```

Arguments

Argument	Description
printer	The printer or class to print to.
num_files	The number of files to print.
files	The files to print.
title	The job title.
num_options	The number of options in the options array.
options	A pointer to the options array.

Returns

The new job ID number or 0 on error.

Description

`cupsPrintFiles()` sends multiple files to the specified printer or class for printing. If the job cannot be printed the error code can be found by calling `cupsLastError()`.

Example

```
#include <cups/cups.h>

int           num_files;
const char    *files[100];
int           num_options;
cups_option_t *options;
int           jobid;

...

jobid = cupsPrintFiles("printer@hostname", num_files, files,
                       "Job Title", num_options, options);
```

See Also

cupsCancelJob(), cupsLastError(), cupsPrintFile()

cupsRasterClose()

Usage

```
void
cupsRasterClose(cups_raster_t *ras);
```

Arguments

Argument	Description
ras	The raster stream to close.

Description

`cupsRasterClose()` closes the specified raster stream.

Example

```
#include <cups/raster.h>

cups_raster_t *ras;

...

cupsRasterClose(ras);
```

See Also

cupsRasterOpen(), cupsRasterReadHeader(), cupsRasterReadPixels(), cupsRasterWriteHeader(), cupsRasterWritePixels()

cupsRasterOpen()

Usage

```
cups_raster_t *
cupsRasterOpen(int fd,
               cups_mode_t mode);
```

Arguments

Argument	Description
fd	The file descriptor to use.
mode	The mode to use; CUPS_RASTER_READ or CUPS_RASTER_WRITE.

Returns

A pointer to a raster stream or NULL if there was an error.

Description

cupsRasterOpen() opens a raster stream for reading or writing.

Example

```
#include <cups/raster.h>

cups_raster_t *ras;

...

ras = cupsRasterOpen(0, CUPS_RASTER_READ);
```

See Also

cupsRasterClose(), cupsRasterReadHeader(), cupsRasterReadPixels(), cupsRasterWriteHeader(), cupsRasterWritePixels()

cupsRasterReadHeader()

Usage

```
unsigned
cupsRasterReadHeader(cups_raster_t *ras,
                     cups_page_header_t *header);
```

Arguments

Argument	Description
ras	The raster stream to read from.
header	A pointer to a page header structure to read into.

Returns

1 on success, 0 on EOF or error.

Description

`cupsRasterReadHeader()` reads a page header from the specified raster stream.

Example

```
#include <cups/raster.h>

int                  line;
cups_raster_t        *ras;
cups_raster_header_t header;
unsigned char        pixels[8192];
...

while (cupsRasterReadHeader(ras, &header))
{
  ...

  for (line = 0; line < header.cupsHeight; line ++)
  {
    cupsRasterReadPixels(ras, pixels, header.cupsBytesPerLine);

    ...
  }
}
```

See Also

cupsRasterClose(), cupsRasterOpen(), cupsRasterReadPixels(), cupsRasterWriteHeader(), cupsRasterWritePixels()

cupsRasterReadPixels()

Usage

```
unsigned
cupsRasterReadPixels(cups_raster_t *ras,
                     unsigned char *pixels,
                     unsigned length);
```

Arguments

Argument	Description
ras	The raster stream to read from.
pixels	The pointer to a pixel buffer.
length	The number of bytes of pixel data to read.

Returns

The number of bytes read or 0 on EOF or error.

Description

`cupsRasterReadPixels()` reads pixel data from the specified raster stream.

Example

```
#include <cups/raster.h>

int                  line;
cups_raster_t        *ras;
cups_raster_header_t header;
unsigned char        pixels[8192];
...

while (cupsRasterReadHeader(ras, &header))
{
  ...

  for (line = 0; line < header.cupsHeight; line ++)
  {
    cupsRasterReadPixels(ras, pixels, header.cupsBytesPerLine);

    ...
  }
}
```

See Also

cupsRasterClose(), cupsRasterOpen(), cupsRasterReadHeader(), cupsRasterWriteHeader(), cupsRasterWritePixels()

cupsRasterWriteHeader()

Usage

```
unsigned
cupsRasterWriteHeader(cups_raster_t *ras,
                      cups_page_header_t *header);
```

Arguments

Argument	Description
ras	The raster stream to write to.
header	A pointer to the page header to write.

Returns

1 on success, 0 on error.

Description

`cupsRasterWriteHeader()` writes the specified page header to a raster stream.

Example

```
#include <cups/raster.h>

int                  line;
cups_raster_t        *ras;
cups_raster_header_t header;
unsigned char        pixels[8192];
...

cupsRasterWriteHeader(ras, &header);

for (line = 0; line < header.cupsHeight; line ++)
{
  ...

  cupsRasterWritePixels(ras, pixels, header.cupsBytesPerLine);
}
```

See Also

cupsRasterClose(), cupsRasterOpen(), cupsRasterReadHeader(), cupsRasterReadPixels(), cupsRasterWritePixels()

cupsRasterWritePixels()

Usage

```
unsigned
cupsRasterWritePixels(cups_raster_t *ras,
                      unsigned char *pixels,
                      unsigned length);
```

Arguments

Argument	Description
ras	The raster stream to write to.
pixels	The pixel data to write.
length	The number of bytes to write.

Returns

The number of bytes written.

Description

`cupsRasterWritePixels()` writes the specified pixel data to a raster stream.

Example

```
#include <cups/raster.h>

int                  line;
cups_raster_t        *ras;
cups_raster_header_t header;
unsigned char        pixels[8192];
...

cupsRasterWriteHeader(ras, &header);

for (line = 0; line < header.cupsHeight; line ++)
{
  ...

  cupsRasterWritePixels(ras, pixels, header.cupsBytesPerLine);
}
```

See Also

cupsRasterClose(), cupsRasterOpen(), cupsRasterReadHeader(), cupsRasterReadPixels(), cupsRasterWriteHeader()

cupsServer()

Usage

```
const char *
cupsServer(void);
```

Returns

A pointer to the default server name.

Description

`cupsServer()` returns a pointer to the default server name. The server name is stored in a static location and will be overwritten with every call to `cupsServer()`

The default server is determined from the following locations:

1. The `CUPS_SERVER` environment variable,
2. The `ServerName` directive in the *client.conf* file,
3. The default host, "localhost".

Example

```
#include <cups/cups.h>

const char *server;

server = cupsServer();
```

See Also

cupsGetPassword(), cupsSetPasswordCB(), cupsSetServer(), cupsSetUser(), cupsUser()

cupsSetPasswordCB()

Usage

```
void
cupsSetPasswordCB(const char *(*cb)(const char *prompt));
```

Arguments

Argument	Description
cb	The password callback function.

Description

`cupsSetPasswordCB()` sets the callback function to use when asking the user for a password. The callback function must accept a single character string pointer (the prompt string) and return `NULL` if the user did not enter a password string or a pointer to the password string otherwise.

Example

```
#include <cups/cups.h>

const char *
my_password_cb(const char *prompt)
{
  return (getpass(prompt));
}

...

char *password;

...

cupsSetPasswordCB(my_password_cb);
password = cupsGetPassword("Please enter a password:");
```

See Also

cupsServer(), cupsSetServer(), cupsSetUser(), cupsUser()

cupsSetServer()

Usage

```
void
cupsSetServer(const char *server);
```

Arguments

Argument	Description
server	The default server to use.

Description

`cupsSetServer()` sets the default server to use for the CUPS API. If the `server` argument is `NULL`, the default server is used.

Example

```
#include <cups/cups.h>

cupsSetServer("foo.bar.com");
```

See Also

cupsServer(), cupsSetPasswordCB(), cupsSetUser(), cupsUser()

cupsSetUser()

Usage

```
void
cupsSetUser(const char *user);
```

Arguments

Argument	Description
user	The user name string to use.

Description

`cupsSetUser()` sets the default user name for authentication. If the `user` argument is `NULL` then the current login user is used.

Example

```
#include <cups/cups.h>

...

cupsSetUser("root");
```

See Also

cupsServer(), cupsSetPasswordCB(), cupsSetServer(), cupsUser()

cupsTempFile()

Usage

```
char *
cupsTempFile(char *filename,
             int length);
```

Arguments

Argument	Description
filename	The character string to hold the temporary filename.
length	The size of the filename string in bytes.

Returns

A pointer to `filename`.

Description

`cupsTempFile()` generates a temporary filename for the */var/tmp* directory or the directory specified by the `TMPDIR` environment variable.

Example

```
#include <cups/cups.h>

char filename[256];

cupsTempFile(filename, sizeof(filename));
```

cupsUser()

Usage

```
const char *
cupsUser(void);
```

Returns

A pointer to the current username or `NULL` if the user ID is undefined.

Description

`cupsUser()` returns the name associated with the current user ID as reported by the `getuid()` system call.

Example

```
#include <cups/cups.h>

const char *user;

user = cupsUser();
```

See Also

cupsGetPassword(), cupsServer()

httpBlocking()

Usage

```
void httpBlocking(http_t *http, int blocking)
```

Arguments

Argument	Description
http	The HTTP connection
blocking	0 if the connection should be non−blocking, 1 if it should be blocking

Description

The `httpBlocking()` function sets the blocking mode for the HTTP connection. By default HTTP connections will block (stop) the client program until data is available or can be sent to the server.

Example

```
#include <cups/http.h>

http_t *http;

http = httpConnect("server", port);
httpBlocking(http, 0);
```

See Also

`httpCheck()`, `httpConnect()`

httpCheck()

Usage

```
int httpCheck(http_t *http);
```

Arguments

Argument	Description
http	The HTTP connection

Returns

0 if there is no data pending, 1 otherwise.

Description

The `httpCheck()` function checks to see if there is any data pending on an HTTP connection.

Example

```
#include <cups/http.h>

http_t *http;

if (httpCheck(http))
{
  ... do something ...
}
```

See Also

`httpBlocking()`, `httpConnect()`, `httpGets()`, `httpRead()`

httpClearFields()

Usage

```
void httpClearFields(http_t *http)
```

Arguments

Argument	Description
http	The HTTP connection

Description

The `httpClearFields()` function clears all HTTP request fields for the HTTP connection.

Example

```
#include <cups/http.h>

http_t *http;

httpClearFields(http);
```

See Also

`httpConnect()`, `httpGetField()`, `httpSetField()`

httpClose()

Usage

```
void httpClose(http_t *http);
```

Arguments

Argument	Description
http	The HTTP connection

Description

The `httpClose()` function closes an active HTTP connection.

Example

```
#include <cups/http.h>

http_t *http;

httpClose(http);
```

See Also

`httpConnect()`

httpConnect()

Usage

```
http_t *httpConnect(const char *hostname, int port);
```

Arguments

Argument	Description
hostname	The name or IP address of the server to connect to
port	The port number to use

Returns

A pointer to a HTTP connection structure or NULL if the connection could not be made.

Description

The `httpConnect()` function opens a HTTP connection to the specified server and port.

Example

```
#include <cups/http.h>

http_t *http;

http = httpConnect(cupsServer(), ippPort());
```

See Also

`httpClose()`, `httpGet()`, `httpGets()`, `httpPost()`, `httpRead()`, `httpWrite()`

httpDecode64()

Usage

```
char *httpDecode64(char *out, const char *in);
```

Arguments

Argument	Description
out	The output string
in	The input string

Returns

A pointer to the decoded string.

Description

The `httpDecode64()` function decodes a base−64 encoded string to the original string.

Example

```
#include <cups/http.h>

char encoded_string[255];
char original_string[255];

httpDecode64(original_string, encoded_string);
```

See Also

`httpEncode64()`

httpDelete()

Usage

```
int httpDelete(http_t *http, const char *uri);
```

Arguments

Argument	Description
http	The HTTP connection
uri	The URI to delete

Returns

0 on success, non−zero on failure.

Description

The `httpDelete()` function sends a HTTP DELETE request to the server.

Example

```
#include <cups/http.h>

http_t *http;

httpDelete(http, "/some/uri");
```

See Also

`httpConnect()`, `httpSetField()`, `httpUpdate()`

httpEncode64()

Usage

```
char *httpEncode64(char *out, const char *in);
```

Arguments

Argument	Description
out	The output string
in	The input string

Returns

A pointer to the encoded string.

Description

The `httpEncode64()` function decodes a base−64 encoded string to the original string.

Example

```
#include <cups/http.h>

char encoded_string[255];
char original_string[255];

httpEncode64(encoded_string, original_string);
```

See Also

`httpDecode64()`

httpError()

Usage

```
int httpError(http_t *http);
```

Arguments

Argument	Description
http	The HTTP connection

Returns

The last error that occurred or 0 if no error has occurred.

Description

The `httpError()` function returns the last error that occurred on the HTTP connection.

Example

```
#include <cups/http.h>

http_t *http;

if (httpError(http))
{
  ... show an error message ...
}
```

See Also

`httpConnect()`

httpFlush()

Usage

```
void httpFlush(http_t *http);
```

Arguments

Argument	Description
http	The HTTP connection

Description

The `httpFlush()` function flushes any remaining data left from a GET or POST operation.

Example

```
#include <cups/http.h>

http_t *http;

httpFlush(http);
```

See Also

`httpConnect()`,

httpGet()

Usage

```
int httpGet(http_t *http, const char *uri);
```

Arguments

Argument	Description
http	The HTTP connection
uri	The URI to get

Returns

0 on success, non−zero on failure.

Description

The `httpGet()` function sends a HTTP GET request to the server.

Example

```
#include <cups/http.h>

http_t *http;

httpGet(http, "/some/uri");
```

See Also

`httpConnect()`, `httpSetField()`, `httpUpdate()`

httpGets()

Usage

```
char *httpGets(char *line, int length, http_t *http)
```

Arguments

Argument	Description
line	The string to fill with a line from the HTTP connection
length	The maximum length of the string
http	The HTTP connection

Returns

A pointer to the string or NULL if no line could be retrieved.

Description

The `httpGets()` function is used to read a request line from the HTTP connection. It is not normally used by a client program.

Example

```
#include <cups/http.h>

http_t *http;
char line[1024];

if (httpGets(line, sizeof(line), http))
{
  ... process the line ...
}
```

See Also

`httpConnect()`, `httpUpdate()`

httpGetDateString()

Usage

```
const char *httpGetDateString(time_t time)
```

Arguments

Argument	Description
time	The UNIX date/time value

Returns

A pointer to a static string containing the HTTP date/time string for the specified UNIX time value.

Description

The `httpGetDateString()` function generates a date/time string suitable for HTTP requests from a UNIX time value.

Example

```
#include <cups/http.h>

puts(httpGetDateString(time(NULL)));
```

See Also

`httpGetDateTime()`

httpGetDateTime()

Usage

```
time_t httpGetDateTime(const char *date)
```

Arguments

Argument	Description
date	The HTTP date/time string

Returns

A UNIX time value.

Description

The `httpGetDateTime()` function converts a HTTP date/time string to a UNIX time value.

Example

```
#include <cups/http.h>

printf("%d\n", httpGetDateTime("Fri, 30 June 2000 12:34:56 GMT"));
```

See Also

`httpGetDateString()`

httpGetField()

Usage

```
const char *httpGetField(http_t *http, http_field_t field);
```

Arguments

Argument	Description
http	The HTTP connection
field	The HTTP field

Returns

A pointer to the field value string.

Description

The `httpGetField()` function returns the current value for the specified HTTP field.

Example

```
#include <cups/http.h>

http_t *http;

httpGet(http, "/some/uri");
while (httpUpdate(http) == HTTP_CONTINUE);

puts(httpGetField(http, HTTP_FIELD_CONTENT_TYPE));
```

See Also

`httpConnect()`, `httpSetField()`

httpHead()

Usage

```
int httpHead(http_t *http, const char *uri);
```

Arguments

Argument	Description
http	The HTTP connection
uri	The URI to head

Returns

0 on success, non−zero on failure.

Description

The `httpHead()` function sends a HTTP HEAD request to the server.

Example

```
#include <cups/http.h>

http_t *http;

httpHead(http, "/some/uri");
```

See Also

`httpConnect()`, `httpSetField()`, `httpUpdate()`

httpInitialize()

Usage

```
void httpInitialize(void);
```

Description

The `httpInitialize()` function initializes the networking code as needed by the underlying platform. It is called automatically by the `httpConnect()` function.

Example

```
#include <cups/http.h>

httpInitialize();
```

See Also

`httpConnect()`

httpOptions()

Usage

```
int httpOptions(http_t *http, const char *uri);
```

Arguments

Argument	Description
http	The HTTP connection
uri	The URI to check for options

Returns

0 on success, non−zero on failure.

Description

The `httpOptions()` function sends a HTTP OPTIONS request to the server.

Example

```
#include <cups/http.h>

http_t *http;

httpOptions(http, "/some/uri");
```

See Also

`httpConnect()`, `httpSetField()`, `httpUpdate()`

httpPost()

Usage

```
int httpPost(http_t *http, const char *uri);
```

Arguments

Argument	Description
http	The HTTP connection
uri	The URI to post to

Returns

0 on success, non−zero on failure.

Description

The `httpPost()` function sends a HTTP POST request to the server.

Example

```
#include <cups/http.h>

http_t *http;

httpPost(http, "/some/uri");
```

See Also

`httpConnect()`, `httpSetField()`, `httpUpdate()`

httpPrintf()

Usage

```
int httpPrintf(http_t *http, const char *format, ...);
```

Arguments

Argument	Description
http	The HTTP connection
format	A printf−style format string

Returns

The number of bytes written.

Description

The `httpPrintf()` function sends a formatted string to the HTTP connection. It is normally only used by the CUPS API and scheduler.

Example

```
#include <cups/http.h>

http_t *http;

httpPrintf(http, "GET / HTTP/1.1 \r\n");
```

See Also

`httpConnect()`

httpPut()

Usage

```
int httpPut(http_t *http, const char *uri);
```

Arguments

Argument	Description
http	The HTTP connection
uri	The URI to put

Returns

0 on success, non−zero on failure.

Description

The `httpPut()` function sends a HTTP PUT request to the server.

Example

```
#include <cups/http.h>

http_t *http;

httpDelete(http, "/some/uri");
```

See Also

`httpConnect()`, `httpSetField()`, `httpUpdate()`

httpRead()

Usage

```
int httpRead(http_t *http, char *buffer, int length);
```

Arguments

Argument	Description
http	The HTTP connection
buffer	The buffer to read into
length	The number of bytes to read

Returns

The number of bytes read or −1 on error.

Description

The `httpRead()` function reads data from the HTTP connection, possibly the result of a GET or POST request.

Example

```
#include <cups/http.h>

http_t *http;
char buffer[1024];
int  bytes;

httpGet(http, "/");
while (httpUpdate(http) != HTTP_CONTINUE);
while ((bytes = httpRead(http, buffer, sizeof(buffer) - 1)) > 0)
{
  buffer[bytes] = '\0';
  fputs(buffer, stdout);
}
```

See Also

`httpConnect()`, `httpWrite()`

httpReconnect()

Usage

```
int httpReconnect(http_t *http);
```

Arguments

Argument	Description
http	The HTTP connection

Returns

0 on success, non−zero on failure.

Description

The `httpReconnect()` function reconnects to the HTTP server. This is usually done automatically if the HTTP functions detect that the server connection has terminated.

Example

```
#include <cups/http.h>

http_t *http;

httpReconnect(http);
```

See Also

`httpConnect()`

httpSeparate()

Usage

```
void httpSeparate(const char *uri, char *method,
                  char *username, char *host, int *port,
                  char *resource);
```

Arguments

Argument	Description
uri	The URI to separate
method	The method (scheme) of the URI
username	The username (and password) portion of the URI, if any
host	The hostname portion of the URI, if any
port	The port number for the URI, either as specified or as default for the method/scheme
resource	The resource string, usually a filename on the server

Description

The `httpSeparate()` function separates the specified URI into its component parts. The method, username, hostname, and resource strings should be at least `HTTP_MAX_URI` characters long to avoid potential buffer overflow problems.

Example

```
char uri[HTTP_MAX_URI];
char method[HTTP_MAX_URI];
char username[HTTP_MAX_URI];
char host[HTTP_MAX_URI];
char resource[HTTP_MAX_URI];
int port;

httpSeparate(uri, method, username, host, &port, resource);
```

See Also

`httpConnect()`

httpSetField()

Usage

```
void httpSetField(http_t *http, http_field_t field, const char *value);
```

Arguments

Argument	Description
http	The HTTP connection
field	The HTTP field
value	The string value for the field

Description

The `httpSetField()` function sets the current value for the specified HTTP field.

Example

```
#include <cups/http.h>

http_t *http;

httpSetField(http, HTTP_FIELD_AUTHORIZATION, "Basic dfdr34453454325"));
httpGet(http, "/some/uri");
while (httpUpdate(http) == HTTP_CONTINUE);
```

See Also

`httpConnect()`, `httpGetField()`

httpTrace()

Usage

```
int httpTrace(http_t *http, const char *uri);
```

Arguments

Argument	Description
http	The HTTP connection
uri	The URI to trace

Returns

0 on success, non−zero on failure.

Description

The `httpTrace()` function sends a HTTP TRACE request to the server.

Example

```
#include <cups/http.h>

http_t *http;

httpTrace(http, "/some/uri");
```

See Also

`httpConnect()`, `httpSetField()`, `httpUpdate()`

httpUpdate()

Usage

```
http_status_t httpUpdate(http_t *http);
```

Arguments

Argument	Description
http	The HTTP connection

Returns

The HTTP status of the current request.

Description

The `httpUpdate()` function updates the current request status. It is used after any DELETE, GET, HEAD, OPTIONS, POST, PUT, or TRACE request to finalize the HTTP request and retrieve the request status.

Since proxies and the current blocking mode can cause the request to take longer, programs should continue calling `httpUpdate()` until the return status is not the constant value `HTTP_CONTINUE`.

Example

```
#include <cups/http.h>

http_t *http;
http_status_t status;

httpGet(http, "/some/uri");
while ((status = httpUpdate(http)) == HTTP_CONTINUE);
printf("Request status is %d\n", status);
```

See Also

`httpConnect()`, `httpDelete()`, `httpGet()`, `httpHead()`, `httpOptions()`, `httpPost()`, `httpPut()`, `httpTrace()`

httpWrite()

Usage

```
int httpWrite(http_t *http, char *buffer, int length);
```

Arguments

Argument	Description
http	The HTTP connection
buffer	The buffer to read into
length	The number of bytes to read

Returns

The number of bytes read or −1 on error.

Description

The `httpWrite()` function reads data from the HTTP connection, possibly the result of a GET or POST request.

Example

```
#include <cups/http.h>

http_t *http;
FILE *fp;
char buffer[1024];
int  bytes;

httpPost(http, "/");

while ((bytes = fread(buffer, 1, sizeof(buffer), fp)) > 0)
  httpWrite(http, buffer, bytes);

while (httpUpdate(http) != HTTP_CONTINUE);

while ((bytes = httpRead(http, buffer, sizeof(buffer) - 1)) > 0)
{
  buffer[bytes] = '\0';
  fputs(buffer, stdout);
}
```

See Also

`httpConnect()`, `httpRead()`

ippAddBoolean()

Usage

```
ipp_attribute_t *ippAddBoolean(ipp_t *ipp, ipp_tag_t group,
                               const char *name, char value);
```

Arguments

Argument	Description
ipp	The IPP request
group	The IPP group
name	The name of attribute
value	The boolean value

Returns

A pointer to the new attribute or NULL if the attribute could not be created.

Description

The `ippAddBoolean()` function adds a single boolean attribute value to the specified IPP request.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;

ippAddBoolean(ipp, IPP_TAG_OPERATION, "my-jobs", 1);
```

See Also

`ippAddBooleans()`, `ippAddDate()`, `ippAddInteger()`, `ippAddIntegers()`, `ippAddRange()`, `ippAddRanges()`, `ippAddResolution()`, `ippAddResolutions()`, `ippAddSeparator()`, `ippAddString()`, `ippAddStrings()`

ippAddBooleans()

Usage

```
ipp_attribute_t *ippAddBooleans(ipp_t *ipp, ipp_tag_t group,
                                const char *name, int num_values,
                                const char *values);
```

Arguments

Argument	Description
ipp	The IPP request
group	The IPP group
name	The name of attribute
num_values	The number of values
values	The boolean values

Returns

A pointer to the new attribute or NULL if the attribute could not be created.

Description

The `ippAddBooleans()` function adds one or more boolean attribute values to the specified IPP request. If the `values` pointer is `NULL` then an array of `num_values` false values is created.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;
char values[10];

ippAddBooleans(ipp, IPP_TAG_OPERATION, "some-attribute", 10, values);
```

See Also

`ippAddBoolean()`, `ippAddDate()`, `ippAddInteger()`, `ippAddIntegers()`, `ippAddRange()`, `ippAddRanges()`, `ippAddResolution()`, `ippAddResolutions()`, `ippAddSeparator()`, `ippAddString()`, `ippAddStrings()`

ippAddDate()

Usage

```
ipp_attribute_t *ippAddDate(ipp_t *ipp, ipp_tag_t group,
                            const char *name, ipp_uchar_t *value);
```

Arguments

Argument	Description
ipp	The IPP request
group	The IPP group
name	The name of attribute
value	The date value

Returns

A pointer to the new attribute or NULL if the attribute could not be created.

Description

The `ippAddDate()` function adds a single date−time attribute value to the specified IPP request.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;

ippAddDate(ipp, IPP_TAG_OPERATION, "some-attribute",
           ippTimeToDate(time(NULL));
```

See Also

`ippAddBoolean()`, `ippAddBooleans()`, `ippAddInteger()`, `ippAddIntegers()`, `ippAddRange()`, `ippAddRanges()`, `ippAddResolution()`, `ippAddResolutions()`, `ippAddSeparator()`, `ippAddString()`, `ippAddStrings()`, `ippTimeToDate()`

ippAddInteger()

Usage

```
ipp_attribute_t *ippAddInteger(ipp_t *ipp, ipp_tag_t group,
                               ipp_tag_t tag, const char *name,
                               int value);
```

Arguments

Argument	Description
ipp	The IPP request
group	The IPP group
tag	The type of integer value (IPP_TAG_INTEGER or IPP_TAG_ENUM)
name	The name of attribute
value	The integer value

Returns

A pointer to the new attribute or NULL if the attribute could not be created.

Description

The `ippAddInteger()` function adds a single integer attribute value to the specified IPP request.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;

ippAddInteger(ipp, IPP_TAG_OPERATION, "limit", 100);
```

See Also

`ippAddBoolean()`, `ippAddBooleans()`, `ippAddDate()`, `ippAddIntegers()`, `ippAddRange()`, `ippAddRanges()`, `ippAddResolution()`, `ippAddResolutions()`, `ippAddSeparator()`, `ippAddString()`, `ippAddStrings()`

ippAddIntegers()

Usage

```
ipp_attribute_t *ippAddIntegers(ipp_t *ipp, ipp_tag_t group,
                                ipp_tag_t tag, const char *name,
                                int num_values, const int *values);
```

Arguments

Argument	Description
ipp	The IPP request
group	The IPP group
tag	The type of integer value (IPP_TAG_INTEGER or IPP_TAG_ENUM)
name	The name of attribute
num_values	The number of values
values	The integer values

Returns

A pointer to the new attribute or NULL if the attribute could not be created.

Description

The `ippAddIntegers()` function adds one or more integer attribute values to the specified IPP request. If the `values` pointer is `NULL` then an array of `num_values` 0 values is created.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;
int values[100];

ippAddIntegers(ipp, IPP_TAG_OPERATION, "some-attribute", 100, values);
```

See Also

ippAddBoolean(), ippAddBooleans(), ippAddDate(), ippAddInteger(), ippAddRange(), ippAddRanges(), ippAddResolution(), ippAddResolutions(), ippAddSeparator(), ippAddString(), ippAddStrings()

ippAddRange()

Usage

```
ipp_attribute_t *ippAddRange(ipp_t *ipp, ipp_tag_t group,
                             const char *name, int low,
                             int high);
```

Arguments

Argument	Description
ipp	The IPP request
group	The IPP group
name	The name of attribute
low	The lower value
high	The higher value

Returns

A pointer to the new attribute or NULL if the attribute could not be created.

Description

The `ippAddRange()` function adds a single range attribute value to the specified IPP request.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;

ippAddRange(ipp, IPP_TAG_OPERATION, "page-ranges", 1, 10);
```

See Also

`ippAddBoolean()`, `ippAddBooleans()`, `ippAddDate()`, `ippAddInteger()`, `ippAddIntegers()`, `ippAddRanges()`, `ippAddResolution()`, `ippAddResolutions()`, `ippAddSeparator()`, `ippAddString()`, `ippAddStrings()`

ippAddRanges()

Usage

```
ipp_attribute_t *ippAddRanges(ipp_t *ipp, ipp_tag_t group,
                              const char *name, int num_values,
                              const int *lows, const int *highs);
```

Arguments

Argument	Description
ipp	The IPP request
group	The IPP group
name	The name of attribute
num_values	The number of range values
lows	The lower values
highs	The higher values

Returns

A pointer to the new attribute or NULL if the attribute could not be created.

Description

The `ippAddRanges()` function adds one or more range attribute values to the specified IPP request. If the `values` pointer is `NULL` then an array of `num_values` 0,0 ranges is created.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;
int lows[2];
int highs[2];

ippAddRanges(ipp, IPP_TAG_OPERATION, "page-ranges", 2, lows, highs);
```

See Also

`ippAddBoolean()`, `ippAddBooleans()`, `ippAddDate()`, `ippAddInteger()`, `ippAddIntegers()`, `ippAddRange()`, `ippAddResolution()`, `ippAddResolutions()`, `ippAddSeparator()`, `ippAddString()`, `ippAddStrings()`

ippAddResolution()

Usage

```
ipp_attribute_t *ippAddResolution(ipp_t *ipp, ipp_tag_t group,
                                  const char *name, int xres,
                                  int yres, ipp_res_t units);
```

Arguments

Argument	Description
ipp	The IPP request
group	The IPP group
name	The name of attribute
xres	The horizontal resolution
yres	The vertical resolution
units	The resolution units

Returns

A pointer to the new attribute or NULL if the attribute could not be created.

Description

The `ippAddResolution()` function adds a single resolution attribute value to the specified IPP request.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;

ippAddBoolean(ipp, IPP_TAG_OPERATION, "printer-resolution",
              720, 720, IPP_RES_PER_INCH);
```

See Also

`ippAddBoolean()`, `ippAddBooleans()`, `ippAddDate()`, `ippAddInteger()`, `ippAddIntegers()`, `ippAddRange()`, `ippAddRanges()`, `ippAddResolutions()`, `ippAddSeparator()`, `ippAddString()`, `ippAddStrings()`

ippAddResolutions()

Usage

```
ipp_attribute_t *ippAddResolutions(ipp_t *ipp, ipp_tag_t group,
                                   const char *name, int num_values,
                                   const int *xres, const int *yres,
                                   const ipp_res_t *units);
```

Arguments

Argument	Description
ipp	The IPP request
group	The IPP group
name	The name of attribute
num_values	The number of resolution values
xres	The horizontal resolutions
yres	The vertical resolutions
units	The resolution units

Returns

A pointer to the new attribute or NULL if the attribute could not be created.

Description

The `ippAddResolutions()` function adds one or more resolution attribute values to the specified IPP request. If the `values` pointer is `NULL` then an array of `num_values` 0,0 resolutions is created.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;
int xres[5];
int yres[5];
ipp_res_t units[5];

ippAddBoolean(ipp, IPP_TAG_OPERATION, "printer-resolutions-supported",
              5, xres, yres, units);
```

See Also

`ippAddBoolean()`, `ippAddBooleans()`, `ippAddDate()`, `ippAddInteger()`, `ippAddIntegers()`, `ippAddRange()`, `ippAddRanges()`, `ippAddResolution()`, `ippAddSeparator()`, `ippAddString()`, `ippAddStrings()`

ippAddSeparator()

Usage

```
ipp_attribute_t *ippAddSeparator(ipp_t *ipp);
```

Arguments

Argument	Description
ipp	The IPP request

Returns

A pointer to the new separator or NULL if the separator could not be created.

Description

The `ippAddSeparator()` function adds a group separator to the specified IPP request.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;

ippAddSeparator(ipp);
```

See Also

`ippAddBoolean()`, `ippAddBooleans()`, `ippAddDate()`, `ippAddInteger()`, `ippAddIntegers()`, `ippAddRange()`, `ippAddRanges()`, `ippAddResolution()`, `ippAddResolutions()`, `ippAddString()`, `ippAddStrings()`

ippAddString()

Usage

```
ipp_attribute_t *ippAddString(ipp_t *ipp, ipp_tag_t group,
                              ipp_tag_t tag, const char *name,
                              const char *charset, const char *value);
```

Arguments

Argument	Description
ipp	The IPP request
group	The IPP group
tag	The type of string value
name	The name of attribute
charset	The character set for the string
value	The string value

Returns

A pointer to the new attribute or NULL if the attribute could not be created.

Description

The `ippAddString()` function adds a single string attribute value to the specified IPP request. For `IPP_TAG_NAMELANG` and `IPP_TAG_TEXTLANG` strings, the charset value is provided with the string to identify the string encoding used. Otherwise the charset value is ignored.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;

ippAddString(ipp, IPP_TAG_OPERATION, IPP_TAG_NAME, "job-name",
             NULL, "abc123");
```

See Also

`ippAddBoolean()`, `ippAddBooleans()`, `ippAddDate()`, `ippAddInteger()`, `ippAddIntegers()`, `ippAddRange()`, `ippAddRanges()`, `ippAddResolution()`, `ippAddResolutions()`, `ippAddSeparator()`, `ippAddStrings()`

ippAddStrings()

Usage

```
ipp_attribute_t *ippAddStrings(ipp_t *ipp, ipp_tag_t group,
                               ipp_tag_t tag, const char *name,
                               int num_values, const char *charset,
                               const char **values);
```

Arguments

Argument	Description
ipp	The IPP request
group	The IPP group
tag	The type of string value
name	The name of attribute
num_values	The number of strings
charset	The character set for the strings
values	The string values

Returns

A pointer to the new attribute or NULL if the attribute could not be created.

Description

The `ippAddStrings()` function adds one or more string attribute values to the specified IPP request. For `IPP_TAG_NAMELANG` and `IPP_TAG_TEXTLANG` strings, the charset value is provided with the strings to identify the string encoding used. Otherwise the charset value is ignored. If the `values` pointer is `NULL` then an array of `num_values` NULL strings is created.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;
char *values[2] = { "one", "two" };

ippAddStrings(ipp, IPP_TAG_OPERATION, IPP_TAG_KEYWORD, "attr-name",
              2, NULL, values);
```

See Also

`ippAddBoolean()`, `ippAddBooleans()`, `ippAddDate()`, `ippAddInteger()`, `ippAddIntegers()`, `ippAddRange()`, `ippAddRanges()`, `ippAddResolution()`, `ippAddResolutions()`, `ippAddSeparator()`, `ippAddString()`

ippDateToTime()

Usage

```
time_t ippDateToTime(const ipp_uchar_t date[11]);
```

Arguments

Argument	Description
date	The IPP date−time value

Returns

A UNIX time value.

Description

The `ippDateToTime()` function converts an IPP date−time value to a UNIX time value.

Example

```
#include <cups/ipp.h>

ipp_uchar_t date[11];

printf("UNIX time is %d\n", ippDateToTime(date));
```

See Also

ippTimeToDate()

ippDelete()

Usage

```
void ippDelete(ipp_t *ipp);
```

Arguments

Argument	Description
ipp	The IPP request or response

Description

The `ippDelete()` function deletes all memory used by an IPP request or response.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;

ippDelete(ipp);
```

See Also

ippNew()

ippFindAttribute()

Usage

Arguments

Argument	Description

Returns

Description

Example

See Also

ippLength()

Usage

Arguments

Argument	Description

Returns

Description

Example

See Also

ippNew()

Usage

```
ipp_t *ippNew(void);
```

Returns

A pointer to a new IPP request or response.

Description

The `ippNew()` function creates a new IPP request or response.

Example

```
#include <cups/ipp.h>

ipp_t *ipp;

ipp = ippNew();
```

See Also

ippDelete()

ippPort()

Usage

```
int ippPort(void);
```

Returns

The default TCP/IP port number for IPP requests.

Description

The `ippPort()` function returns the default IPP port number for requests.

Example

```
#include <cups/http.h>
#include <cups/ipp.h>

http_t *http;

http = httpConnect(cupsServer(), ippPort());
```

See Also

`cupsServer()`, `ippSetPort()`

ippRead()

Usage

```
ipp_state_t ippRead(http_t *http, ipp_t *ipp);
```

Arguments

Argument	Description
http	The HTTP connection
ipp	The IPP request or response

Returns

The current read state.

Description

The `ippRead()` function reads IPP attributes from the specified HTTP connection. Programs should continue calling `ippRead()` until `IPP_ERROR` or `IPP_DATA` is returned.

Example

```
#include <cups/http.h>
#include <cups/ipp.h>

http_t *http;
ipp_t *ipp;
ipp_state_t status;

ipp = ippNew();

while ((status = ippRead(http, ipp)) != IPP_ERROR)
  if (status == IPP_DATA)
    break;

if (status == IPP_DATA)
{
  ... read additional non-IPP data using httpRead() ...
}
```

See Also

ippWrite()

ippSetPort()

Usage

```
void
ippSetPort(int port);
```

Arguments

Argument	Description
port	The port number to use

Description

The `ippSetPort()` function sets the default IPP port number for requests.

Example

```
#include <cups/http.h>
#include <cups/ipp.h>

...

ippSetPort(8631);
```

See Also

ippPort()

ippTimeToDate()

Usage

```
ipp_uchar_t *ippTimeToDate(time_t time);
```

Arguments

Argument	Description
time	The UNIX time value

Returns

A static pointer to an IPP date−time value.

Description

The `ippTimeToDate()` function converts a UNIX time to an IPP date−time value.

Example

```
#include <cups/ipp.h>

ipp_uchar_t *date;

date = ippTimeToDate(time(NULL));
```

See Also

ippDateToTime()

ippWrite()

Usage

```
ipp_state_t ippWrite(http_t *http, ipp_t *ipp);
```

Arguments

Argument	Description
http	The HTTP connection
ipp	The IPP request or response

Returns

The current write state.

Description

The `ippWrite()` function writes IPP attributes to the specified HTTP connection. Programs should continue calling `ippWrite()` until `IPP_ERROR` or `IPP_DATA` is returned.

Example

```
#include <cups/http.h>
#include <cups/ipp.h>

http_t *http;
ipp_t *ipp;
ipp_state_t status;

ipp = ippNew();
... add attributes ...

while ((status = ippWrite(http, ipp)) != IPP_ERROR)
  if (status == IPP_DATA)
    break;

if (status == IPP_DATA)
{
  ... read additional non-IPP data using httpWrite() ...
}
```

See Also

ippRead()

ppdClose()

Usage

```
void ppdClose(ppd_file_t *ppd);
```

Arguments

Argument	Description
ppd	The PPD file

Description

The `ppdClose()` function frees all memory associated with the PPD file.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;

ppdClose(ppd);
```

See Also

`ppdOpen()`, `ppdOpenFd()`, `ppdOpenFile()`

ppdConflicts()

Usage

```
int ppdConflicts(ppd_file_t *ppd);
```

Arguments

Argument	Description
ppd	The PPD file

Returns

The number of option conflicts in the file.

Description

The `ppdConflicts()` function returns the number of conflicts with the currently selected options.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;

printf("%d conflicts\n", ppdConflicts(ppd));
```

See Also

`cupsMarkOptions()`, `ppdIsMarked()`, `ppdMarkDefaults()`, `ppdMarkOption()`

ppdEmit()

Usage

```
int ppdEmit(ppd_file_t *ppd, FILE *file, ppd_section_t section);
```

Arguments

Argument	Description
ppd	The PPD file
file	The file to write to
section	The option section to write

Returns

0 on success, −1 on error.

Description

The `ppdEmit()` function sends printer−specific option commands to the specified file.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;

ppdEmit(ppd, stdout, PPD_ORDER_PAGE);
```

See Also

`ppdEmitFd()`

ppdEmitFd()

Usage

```
int ppdEmitFd(ppd_file_t *ppd, int fd, ppd_section_t section);
```

Arguments

Argument	Description
ppd	The PPD file
fd	The file descriptor to write to
section	The option section to write

Returns

0 on success, −1 on error.

Description

The `ppdEmitFd()` function sends printer−specific option commands to the specified file descriptor.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;

ppdEmitFd(ppd, 1, PPD_ORDER_PAGE);
```

See Also

ppdEmit()

ppdFindChoice()

Usage

```
ppd_choice_t *ppdFindChoice(ppd_option_t *option, const char *choice);
```

Arguments

Argument	Description
option	A pointer to the option
choice	The name of the choice

Returns

A pointer to the choice data or NULL if the choice does not exist.

Description

The `ppdFindChoice()` function returns a pointer to the choice data for the specified option.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;
ppd_option_t *option;
ppd_choice_t *choice;

option = ppdFindOption(ppd, "PageSize");
choice = ppdFindChoice(option, "Letter");
```

See Also

`ppdFindMarkedChoice()`, `ppdFindOption()`

ppdFindMarkedChoice()

Usage

```
ppd_choice_t *ppdFindMarkedChoice(ppd_file_t *ppd, const char *keyword);
```

Arguments

Argument	Description
ppd	The PPD file
keyword	The name of the option

Returns

A pointer to the choice data or NULL if the choice does not exist or is not marked.

Description

The `ppdFindMarkedChoice()` function returns a pointer to the marked choice data for the specified option.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;
ppd_choice_t *choice;

choice = ppdFindMarkedChoice(ppd, "PageSize");
```

See Also

`ppdFindChoice()`, `ppdFindOption()`

ppdFindOption()

Usage

```
ppd_option_t *ppdFindOption(ppd_file_t *ppd, const char *keyword);
```

Arguments

Argument	Description
ppd	The PPD file
keyword	The name of the option

Returns

A pointer to the option data or NULL if the option does not exist.

Description

The `ppdFindOption()` function returns a pointer to the option data for the specified option.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;
ppd_option_t *option;

option = ppdFindOption(ppd, "PageSize");
```

See Also

`ppdFindChoice()`, `ppdFindMarkedChoice()`

ppdIsMarked()

Usage

```
int ppdIsMarked(ppd_file_t *ppd, const char *keyword, char char *choice);
```

Arguments

Argument	Description
ppd	The PPD file
keyword	The name of the option
choice	The name of the option choice

Returns

1 if the choice is marked, 0 otherwise.

Description

The `ppdIsMarked()` function returns whether or not the specified option choice is marked.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;

printf("Letter size %s selected.\n",
       ppdIsMarked(ppd, "PageSize", "Letter") ? "is" : "is not");
```

See Also

`cupsMarkOptions()`, `ppdConflicts()`, `ppdIsMarked()`, `ppdMarkDefaults()`, `ppdMarkOption()`

ppdMarkDefaults()

Usage

```
void ppdMarkDefaults(ppd_file_t *ppd);
```

Arguments

Argument	Description
ppd	The PPD file

Description

The `ppdMarkDefaults()` function marks all of the default choices in the PPD file.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;

ppdMarkDefaults(ppd);
```

See Also

`cupsMarkOptions()`, `ppdConflicts()`, `ppdIsMarked()`, `ppdMarkDefaults()`, `ppdMarkOption()`

ppdMarkOption()

Usage

```
int ppdMarkOption(ppd_file_t *ppd, const char *keyword, const char *choice);
```

Arguments

Argument	Description
ppd	The PPD file
keyword	The name of the option
choice	The name of the choice

Returns

The number of conflicts in the PPD file.

Description

The `ppdMarkOption()` function marks the specified option choice.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;

ppdMarkOption(ppd, "PageSize", "Letter");
```

See Also

`cupsMarkOptions()`, `ppdConflicts()`, `ppdIsMarked()`, `ppdMarkDefaults()`, `ppdMarkOption()`

ppdOpen()

Usage

```
ppd_file_t *ppdOpen(FILE *file);
```

Arguments

Argument	Description
file	The file to read from

Returns

A pointer to a PPD file structure or NULL if the PPD file could not be read.

Description

The `ppdOpen()` function reads a PPD file from the specified file into memory.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;
FILE *file;

file = fopen("filename.ppd", "rb");
ppd = ppdOpen(file);
fclose(file);
```

See Also

`ppdClose()`, `ppdOpenFd()`, `ppdOpenFile()`

ppdOpenFd()

Usage

```
ppd_file_t *ppdOpenFd(int fd);
```

Arguments

Argument	Description
fd	The file descriptor to read from

Returns

A pointer to a PPD file structure or NULL if the PPD file could not be read.

Description

The `ppdOpenFd()` function reads a PPD file from the specified file descriptor into memory.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;
int        fd;

fd = open("filename.ppd", O_RDONLY);
ppd = ppdOpenFd(fd);
close(fd);
```

See Also

`ppdClose()`, `ppdOpen()`, `ppdOpenFile()`

ppdOpenFile()

Usage

```
ppd_file_t *ppdOpenFile(const char *filename);
```

Arguments

Argument	Description
filename	The name of the file to read from

Returns

A pointer to a PPD file structure or NULL if the PPD file could not be read.

Description

The `ppdOpenFile()` function reads a PPD file from the named file into memory.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;

ppd = ppdOpenFile("filename.ppd");
```

See Also

`ppdClose()`, `ppdOpen()`, `ppdOpenFd()`

ppdPageLength()

Usage

```
float ppdPageLength(ppd_file_t *ppd, const char *name);
```

Arguments

Argument	Description
ppd	The PPD file
name	The name of the page size

Returns

The length of the specified page size in points or 0 if the page size does not exist.

Description

The `ppdPageLength()` function returns the page length of the specified page size.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;

printf("Length = %.0f\n", ppdPageLength(ppd, "Letter"));
```

See Also

`ppdPageLength()`, `ppdPageSize()`, `ppdPageWidth()`

ppdPageSize()

Usage

```
ppd_size_t *ppdPageSize(ppd_file_t *ppd, const char *name);
```

Arguments

Argument	Description
ppd	The PPD file
name	The name of the page size

Returns

A pointer to the page size record of the specified page size in points or NULL if the page size does not exist.

Description

The `ppdPageSize()` function returns the page size record for the specified page size.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;
ppd_size_t *size;

size = ppdPageSize(ppd, "Letter");
if (size != NULL)
{
  printf(" Width = %.0f\n", size->width);
  printf("Length = %.0f\n", size->length);
  printf("  Left = %.0f\n", size->left);
  printf(" Right = %.0f\n", size->right);
  printf("Bottom = %.0f\n", size->bottom);
  printf("   Top = %.0f\n", size->top);
}
```

See Also

`ppdPageLength()`, `ppdPageWidth()`

ppdPageWidth()

Usage

```
float ppdPageWidth(ppd_file_t *ppd, const char *name);
```

Arguments

Argument	Description
ppd	The PPD file
name	The name of the page size

Returns

The width of the specified page size in points or 0 if the page size does not exist.

Description

The `ppdPageWidth()` function returns the page width of the specified page size.

Example

```
#include <cups/ppd.h>

ppd_file_t *ppd;

printf("Width = %.0f\n", ppdPageWidth(ppd, "Letter"));
```

See Also

`ppdPageLength()`, `ppdPageSize()`